UYGULAMALI ÖĞRETİM SERİSİ

CCS C
İle
PIC Programlama

Serdar ÇİÇEK

2007

Yayın Hakları:
Bu kitabın yayın hakları ALTAŞ YAYINCILIK ve ELEKTRONİK TİC. LTD. ŞTİ'ne aittir. Yayıncının ve yazarlarının izni olmadan hiç bir biçimde ve hiç bir yolla kısmen ya da tümü yeniden basılamaz, taklit edilemez, fotokopi veya benzeri cihazlarla çoğaltılamaz. Bilgisayar ortamına aktarılamaz.

CCS C İle PIC Programlama

Yazar : Serdar ÇİÇEK

Birinci basım **Eylül 2007** İstanbul

Teknik Hizmetler ve Baskı:
Melisa Matbaacılık: (0212) 501 97 57

ALTAŞ Yayıncılık ve Elektronik Tic. Ltd. Şti.
İnönü Caddesi Faikbey Sokak No:11/B
34844 Maltepe/İSTANBUL

Telefon/Faks: (0216) 305 79 46

http:// www.altaskitap.com
Sipariş e-posta: siparis@altaskitap.com

ISBN : 978-975-8834-20-4

ÖNSÖZ

Teknoloji, geçmiş yıllardaki dikey gelişmelere paralel olarak, günümüzde son derece hızlı bir şekilde yatay gelişmeler göstermektedir. Öyle ki bu teknolojik gelişmeler meslek sahibi bir kişinin kendisini geliştirmediği sürece edindiği bilgiler ile sadece 2 yıl idare edebileceği gerçeği ile yüz yüze getirmiştir. Diğer alanlarda olduğu gibi elektronik alanında da hızlı bir gelişme dikkat çekmektedir. Elektronik alanında meydana gelen bu gelişmeler, yeni ürünlerin, cihazların ve sistemlerin kontrolünü karmaşık bir hale getirmiştir. Bu karmaşık sistemleri kontrol etmek için ise mikrodenetleyici ürünü ön plana çıkmaktadır. Mikrodenetleyici ürünlerinin donanımları da teknolojideki bu gelişmeleri kontrol edebilecek düzeyde geliştirilmiştir. Ayrıca mikrodenetleyicilerin donanım yapısının gelişmesinin yanında, mikrodenetleyici yazılım yelpazesi de oldukça büyümüştür. Günümüzde istenen bir denetleyici ürünü birçok orta ve yüksek seviyeli dil ile programlanabilmektedir.

Değişik firmaların farklı özellikte birçok mikrodenetleyici ürünleri bulunmaktadır. Bu ürünler içinde Microchip firmasının PIC mikrodenetleyici ürünleri donanım yapısının zenginliği, ucuz oluşu ve piyasa da kolaylıkla bulunması nedeniyle oldukça popüler olmuştur. Alçak seviyeli dil kategorisinde program yazmada Assembly dili en ideal dildir. Fakat alçak seviyeli dil ile program yazmak insan diline olan uzaklığı nedeniyle hem zor hem de çok dikkat gerektiren bir iştir. Bunun yerine günümüzde insan diline yakın olan orta ve yüksek seviyeli diller ile mikrodenetleyici programlamak oldukça yaygınlaşmıştır. **Elektronik alanında sadece mikrodenetleyici ürünleri arasında değil diğer elektronik ürünler arasında da C dili çok yaygın bir şekilde kullanılmaktadır.**

CCS (**C**ustom **C**omputer **S**ervices Inc.) firması PIC denetleyicilerin C dilinde programlanmasını sağlayan tümleşik yazılımlar geliştiren ve bu alanda deneme kartları üreten bir firmadır. Bu firmanın PIC ürünleri için **CCS C** adında bir C derleyici programı mevcuttur. Bu derleyici diğer C derleyicilerinden farklı olarak neredeyse tümüyle ANSI C dilini desteklemektedir. CCS C programı PIC10XX, PIC12XX, PIC14XX, PIC16XX, PIC18XX ürünlerini desteklemekte ve 24 bit'lik PIC ile dsPIC ürünleri için de ayrı versiyonları bulunmaktadır. CCS C derleyicisi içinde bulunan hazır fonksiyonların yanında çevresel birimler ve iletişim protokolleri için hazır birçok kütüphane dosyası bulunmaktadır. Buna ilaveten CCS C'nin kendi forum sitesi ve diğer sitelerde CCS C için yazılmış yüzlerce kütüphane dosyası rahatlıkla bulunmaktadır. CCS C derleyicisi ayrıca diğer C derleyicilerine nazaran daha kolay bir şekilde programlama yapmaya imkân vermektedir. Kullanılan denetleyicinin donanım yapısını tam bilmeden bile ileri düzeyde programlar yazılabilmektedir. Bu özellik CCS C derleyicisini diğer C derleyicilerine göre daha popüler yapmıştır. Bahsedilen avantajları nedeniyle CCS C derleyicisi tercih edilmiştir. Kitapta PIC16F877 ve PIC16F877A ürünlerinin donanım yapısıyla birlikte CCS C programının tüm yönleri ile kullanılması anlatılmıştır.

Ayrıca kitapta mikrodenetleyici içeren bir sistem tasarımında kullanılabilecek çevresel birimler ile ilgili de teknik bilgi mevcuttur. Kitapta verilen uygulama devrelerinin çoğunluğu Altaş Yayıncılık ve Elektronik şirketinin ürünü olan PIC PROG/DEKA 16F87X deneme kartı ile uyumludur. Ayrıca tüm uygulama devreleri Proteus-ISIS programın da denenmiştir. Kitapta uygulama örneklerinin başlıklarının altında aşağıda verilen şekilde bir çubuk bulunmaktadır.

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu çubukta uygulama devresinin PROTEUS-ISIS benzetim (simülasyon) programında ve PIC PROG/DEKA 16F87X deneme kartında çalışıp çalışmadığı bilgisi verilmektedir. Kutucuklarda verilen ☺ sembolü belirtilen ortamda devrenin çalıştırabileceğini, ☹ sembolü ise çalıştırılamayacağını belirtir. Bazı uygulamalar direkt olarak PIC PROG/DEKA 16F87X deneme kartında çalıştırılamasa da bir board üzerine yerleştirilen ek birkaç eleman vasıtası ile kolayca denenebilir. Bu gibi bilgiler kitapta ilgili kısımlarda verilmiştir.

Kitapta verilen uygulama devrelerinin C ve Hex kodları ile uygulama devrelerinin Proteus-ISIS benzetim dosyaları www.altaskitap.com adresinden indirilebilir.

Gösterilen tüm dikkat ve öneme rağmen kitap anlatımında ve içeriğinde gözden kaçan hatalar ile ilgili eleştiri ve uyarılarınızı iletmeniz bizleri daha da geliştirecektir.

Kitabın tüm ilgilenenlere faydalı olması dileğiyle...

Serdar ÇİÇEK
Ağustos 2007
drserdarc@hotmail.com

Her zaman yanımda olan Annem, Babam ve Kardeşlerime...

İÇİNDEKİLER

BÖLÜM 1: GİRİŞ **11**
1.1. GİRİŞ 11
1.2. MİKRODENETLEYİCİLERE GENEL BİR BAKIŞ 12
1.3. MİKRODENETLEYİCİ PROGRAMLAMAK İÇİN GEREKENLER 13
BÖLÜM 2: PIC16F877 MİKRODENETLEYİCİSİ **15**
2.1. PIC DENETLEYİCİLER VE PIC16F877'NİN GENEL ÖZELLİKLERİ 15
2.2. BELLEK ORGANİZASYONU 19
2.2.1. Program Belleği (Flash Bellek) 19
2.2.2. Veri Belleği (RAM Bellek) 20
2.3. BESLEME UÇLARI VE BAĞLANTILARI 22
2.4. OSİLATÖR KONFİGÜRASYONLARI 22
2.4.1. Kristal Osilatörler 23
2.4.2. R/C Osilatörü 24
2.4.3. Harici Osilatör 25
2.5. SIFIRLAMA (RESET) DEVRESİ VE ÇEŞİTLERİ 25
2.5.1. Besleme İlk Verildiği Anda Sıfırlama - Power On Reset (POR) 26
2.5.2. Güç Zamanlayıcılı Sıfırlama - Power Up Timer (PWRT) 26
2.5.3. Osilatör Başlama Zamanlayıcısı - Oscillator Start-up Timer (OST) 26
2.5.4. Gerilim Düşmesi Sıfırlama - Brown Out Reset (BOR) 26
2.5.5. Watch Dog Timer (WDT) Sıfırlaması 26
2.5.6. $\overline{MCLR}$ Sıfırlaması 27
2.6. GİRİŞ / ÇIKIŞ (I/O) PORTLARI 28
2.6.1. PORT A 28
2.6.2. PORT B 29
2.6.3. PORT C, PORT D ve PORT E 29
2.7. KONFİGÜRASYON BİT'LERİ 31
2.8. KESMELER (INTERRUPTS) 31
2.9. ÜRÜN TANIMLAMA KOD AÇILIMLARI 32
BÖLÜM 3: CCS C DERLEYİCİSİ **33**
3.1. CCS C PROGRAMI 33
3.2. CCS C DEMO PROGRAMININ KURULMASI 35
3.3. CCS C DERLEYİCİSİ MENÜLERİ 38
3.3.1. File Menüsü 38
3.3.2. Project Menüsü Şeridi 39
3.3.3. Edit Menüsü Şeridi 39
3.3.4. Search Menüsü Şeridi 40
3.3.5. Options Menüsü Şeridi 40
3.3.6. Compile Menüsü Şeridi 41
3.3.7. View Menüsü Şeridi 42
3.3.8. Tools Menüsü Şeridi 43
3.3.9. Debug Menüsü Şeridi 44
3.3.10. Document Menüsü Şeridi 44
3.3.11. User ToolBar Menüsü Şeridi 45
3.3.12. Help Menüsü 46
BÖLÜM 4: CCS C DERLEYİCİSİ TEMEL KAVRAMLARI **47**
4.1. TANITICI KELİMELER 48
4.2. C'DE PROGRAM SATIRLARI 48
4.3. DEĞİŞKEN TİPLERİ VE TANIMLAMALARI 49
4.4. YER BELİRLEYİCİLERİ VE TANIMLAMALARI 50
4.5. DEĞİŞKEN TİP İSİMLERİNE TAKMA İSİM VERME (ALIAS) 51
4.6. SABİT TANIMLAMA VE ROM BELLEĞE BİLGİ YAZMA 51

4.6.1. BackSlash (Ters Eğik Çizgi-\) Sabitleri 51
4.6.2. % Sabitleri 52
4.7. CCS C DERLEYİCİSİNDE KULLANILAN OPERATÖRLER 52
4.7.1. Aritmetiksel Operatörler 53
4.7.2. İlişkisel Operatörler 54
4.7.3. Mantıksal Operatörler 55
4.7.4. Bit Operatörleri 55
4.7.4.1. Bit Değil (~) Operatörü 56
4.7.4.2. Bit VE (&) Operatörü 56
4.7.4.3. Bit VEYA (|) Operatörü 57
4.7.4.4. Bit ÖZEL VEYA (^) Operatörü 57
4.7.4.5. Sola Öteleme (<<) Operatörü 57
4.7.4.6. Sağa Öteleme (>>) Operatörü 58
4.7.5. Unary Operatörler 58
4.7.6. Özel Amaçlı Operatörler 58
4.7.6.1. Atama (=) Operatörü 58
4.7.6.2. İşlemli Atama Operatörü 59
4.8. CCS C'DE ÖZEL KARAKTERLERİN GÖSTERİMİ 59
4.9. DİZİ TANIMLAMA 59
4.9.1. Dizilere Değer Verme ve Birden Fazla Boyutlu Diziler 60
4.10. GÖSTERİCİLER (POINTERS) 61
4.10.1. Gösterici Bildirimleri 61
4.10.2. Gösterici Kullanımı 62
4.10.3. Adres Operatörü 64
4.11. YAPILAR (STRUCTURES) 64
4.12. CCS C'DE FONKSİYON TANIMLAMA 66
4.13. AÇIKLAYICI YAZIMI 66
4.14. CCS C PROGRAMI GENEL GÖVDESİ 67
4.15. CCS KOMUT SETİ 69
4.16. ÖN İŞLEMCİ FONKSİYONLARI 71
4.17. MİKRODENETLEYİCİ KONTROL FONKSİYONLARI 78
4.18. KARAKTER FONKSİYONLARI 81
4.19. MATEMATİK FONKSİYONLARI 84
BÖLÜM 5: CCS C İLE PROGRAM DERLEMEK VE DENEMEK 87
5.1. CCS C'DE DERLEME İŞLEMİ 87
5.1.1. Otomatik Olarak Proje Dosyası Oluşturmak ve Derlemek 87
5.1.2. El İle (Manuel) Olarak Proje Dosyası Oluşturmak Ve Derlemek 93
5.2. DERLEME SONUCU OLUŞAN VE CCS C'NİN TANIDIĞI DOSYA FORMATLARI 95
5.3. PIC PROG/DEKA 16F87X DENEME KARTI 95
5.3.1. PIC PROG/DEKA Deneme Kartı Özellikleri 96
5.3.2. Deneme Kartının Birimleri 97
5.3.2.1. A Portu 98
5.3.2.2. B Portu 98
5.3.2.3. C Portu 99
5.3.2.4. D Portu 100
5.3.2.5. E Portu 100
5.3.2.6. Deneme Kartı Osilatör Birimi 101
5.4. PIC'E PROGRAM YÜKLEMEK (IC-PROG İLE) 101
BÖLÜM 6: PORT GİRİŞ/ÇIKIŞ VE BİT/BYTE İŞLEMLERİ 105
6.1. PORT GİRİŞ/ÇIKIŞ KOMUTLARI 105
6.1.1. SET_TRIS_X() Komutu 105
6.1.2. GET_TRIS_X() Komutu 106
6.1.3. OUTPUT_LOW() Komutu 106
6.1.4. OUTPUT_HIGH() Komutu 106
6.1.5. OUTPUT_BIT() Komutu 107

6.1.6. OUTPUT_X() Komutu 107
6.1.7. OUTPUT_TOGGLE() Komutu 107
6.1.8. OUTPUT_FLOAT() Komutu 107
6.1.9. INPUT() Komutu 108
6.1.10. INPUT_X() Komutu 108
6.1.11. INPUT_STATE() 108
6.1.12. PORT_A_PULLUPS(), PORT_B_PULLUPS() Komutları 108
6.2. BİT VE BYTE İŞLEMLERİ KOMUTLARI 109
6.2.1. BIT_SET() Komutu 109
6.2.2. BIT_CLEAR() Komutu 109
6.2.3. BIT_TEST() Komutu 109
6.2.4. SWAP() Komutu 110
6.2.5. MAKE8() Komutu 110
6.2.6. MAKE16() Komutu 110
6.2.7. MAKE32() Komutu 111
6.3. LED YAKMA VE SÖNDÜRME UYGULAMASI 111
6.4. BUTON İLE LED KONTROL UYGULAMASI 115
6.5. LED TOGGLE UYGULAMASI 117
6.6. PORT OKUMA VE PORT'A BİLGİ GÖNDERME UYGULAMASI 119
BÖLÜM 7: KARAR VERME DÖNGÜ İŞLEMLERİ VE GECİKME FONKSİYONLARI 123
7.1. KARAR VERME DÖNGÜ DEYİMLERİ 123
7.1.1. "if" Deyimi 124
7.1.2. "if else" Deyimi 124
7.1.3. "switch-case" Şartlı Dallanma Deyimi 125
7.1.4. "for" Döngüsü 125
7.1.5. "while" döngüsü 126
7.1.6. "do-while" Döngüsü 127
7.1.7. "continue" Komutu 127
7.1.8. "goto" Komutu 127
7.2. GECİKME FONKSİYONLARI 127
7.2.1. DELAY_US() Fonksiyonu 128
7.2.2. DELAY_MS() Fonksiyonu 128
7.2.3. DELAY_CYCLES() Fonksiyonu 128
7.3. BUTON İLE BCD SAYICI UYGULAMASI 128
7.4. YÜRÜYEN IŞIK UYGULAMASI 131
7.5. DISPLAY UYGULAMASI-1 (0-F SAYICI) 134
7.6. DİSPLAY UYGULAMASI-2 (BUTON İLE 0-99 YUKARI SAYICI) 138
7.7. DISPLAY UYGULAMASI-3 (7448 ENTEGRESİ VE BUTON İLE YUKARI-AŞAĞI 0-99 SAYICI) 143
BÖLÜM 8: PORT ÇOĞULLAMA VE MATRİS LED DİSPLAY UYGULAMALARI 149
8.1. 74164 ENTEGRESİ 149
8.2. 74138 ENTEGRESİ 151
8.3. 8255 ENTEGRESİ 153
8.3.1. 8255 Çalışma Modları ve Kontrol Kaydedicisi 155
8.4. 74164 İLE 2 ÇIKIŞLA 7 SEGMENT DİSPLAY SÜRMEK 156
8.5. 74138 İLE PORT ÇOĞULLAMA UYGULAMASI 159
8.6. 8255 İLE PORT ÇOĞULLAMA UYGULAMASI 161
8.7. MATRİS LED DİSPLAY 165
8.8. MATRİS LED DİSPLAY UYGULAMASI 168
BÖLÜM 9: KESME (INTERRUPT) İŞLEMLERİ 173
9.1. KESME MANTIĞI VE KESME OLAYLARINA GENEL BİR BAKIŞ 173
9.2. CCS C'DE KESME OLUŞTURMA İŞLEMLERİ 180
9.3. DIŞ KESME (EXTERNAL INTERRUPT-EXT_INT) UYGULAMASI 182
9.4. B PORT'UNDAKİ B4...B7 PİN'LERİNDE DEĞİŞİKLİK KESMESİ UYGULAMASI 184
BÖLÜM 10: ZAMANLAYICI VE SAYICI İŞLEMLERİ 187
10.1. ZAMANLAYICI/SAYICI KAVRAMLARI 187

10.2. TIMER0 BİRİMİ 188
10.2.1. SETUP_TIMER_0 () Fonksiyonu 190
10.2.2. SET_TIMER0 ve SET_RTCC() Fonksiyonu 191
10.2.3. Timer0 Kesmesi (#INT_TIMER0) 191
10.2.4. GET_TIMERx () Fonksiyonu 192
10.2.5. Timer0 Uygulaması-1 192
10.2.6. Timer0 Uygulaması-2 195
10.3. TIMER1 BİRİMİ 197
10.3.1. Timer1 Kesmesi (#INT_TIMER1) 199
10.3.2. SETUP_TIMER1() Fonksiyonu 199
10.3.3. SET_TIMER1() Fonksiyonu 200
10.3.4. Timer1 Uygulaması-1 200
10.3.5. Timer1 Uygulaması-2 203
10.4. TIMER2 BİRİMİ 205
10.4.1. Timer2 Kesmesi (#INT_TIMER2) 207
10.4.2. SETUP_TIMER_2() Fonksiyonu 207
10.4.3. SET_TIMER2() Fonksiyonu 208
10.4.4. Timer2 Uygulaması 208
10.5. WATCHDOG TIMER (WDT) BİRİMİ 210
10.5.1. SETUP_WDT() Fonksiyonu 212
10.5.2. RESTART_WDT() Fonksiyonu 213
10.5.3. WDT Uygulaması 213
BÖLÜM 11: CAPTURE/COMPARE/PWM BİRİMİ İŞLEMLERİ 217
11.1. CCP BİRİMİNE GENEL BAKIŞ 217
11.2. CAPTURE MODU 219
11.3. COMPARE MODU 220
11.4. PWM MODU 220
11.5. SETUP_CCPX() FONKSİYONU 222
11.6. SET_PWMX_DUTY() FONKSİYONU 223
11.7. CAPTURE MODU UYGULAMASI 224
11.8. COMPARE MODU UYGULAMASI 227
11.9. PWM MODU UYGULAMASI-1 231
11.10. PWM MODU UYGULAMASI-2 234
BÖLÜM 12: TUŞ TAKIMI (KEYPAD) UYGULAMALARI 237
12.1. TUŞ TAKIMI (KEYPAD) HAKKINDA BİLGİ 237
12.2. TUŞ TAKIMI UYGULAMASI-1 239
12.3. 74C922 KEY ENKODER ENTEGRESİ 243
12.4. TUŞ TAKIMI UYGULAMASI-2 (74C922 İLE) 245
BÖLÜM 13: KARAKTER LCD VE GRAFİK LCD UYGULAMALARI 247
13.1. PARALEL LCD 249
13.1.1. LCD DDRAM Belleği ve LCD Karakter Tablosu 252
13.1.2. HD44780 Tabanlı LCD komutları 254
13.1.3. Paralel LCD'yi Direkt Bilgisayara Bağlamak 258
13.2. SERİ LCD 258
13.2.1. Seri LCD Komutları ve DDRAM Adresleri 259
13.2.2. Seri LCD'yi Direkt Bilgisayara Bağlamak 260
13.3. CCS C KARAKTER TABANLI PARALEL LCD KÜTÜPHANESİ (LCD.C VE LCD420.C) 261
13.4. PARALEL LCD UYGULAMASI 263
13.5. TUŞ TAKIMI VE LCD UYGULAMASI 266
13.6. KENDİ LCD DOSYAMIZI OLUŞTURMA 269
13.7. KENDİ LCD DOSYAMIZ İLE LCD UYGULAMASI 273
13.8. LCD'DE TÜRKÇE VE ÖZEL KARAKTER KULLANMAK 275
13.9. ÖZEL KARAKTERLİ LCD DOSYASI OLUŞTURMAK 277
13.10. ÖZEL KARAKTERLİ LCD UYGULAMASI 281
13.11. GRAFİK LCD 282

13.11.1. CCS C Grafik LCD Kütüphane Dosyaları 283
13.12. GRAFİK LCD UYGULAMASI 286
BÖLÜM 14: ANALOG/DİJİTAL ÇEVİRİCİ MODÜLÜ İŞLEMLERİ 291
14.1. A/D MODÜLÜ 291
14.1.1. A/D Kesmesi 295
14.2. SETUP_ADC() FONKSİYONU 295
14.3. SETUP_ADC_PORTS() 296
14.4. SET_ADC_CHANNEL() FONKSİYONU 297
14.5. READ_ADC() FONKSİYONU 298
14.6. #DEVICE ADC=X KOMUTU 298
14.7. ADC UYGULAMASI 299
14.8. LM35 SICAKLIK SENSÖRÜ 304
14.9. LM35 SICAKLIK SENSÖRÜ UYGULAMASI 305
BÖLÜM 15: DAHİLİ EEPROM İŞLEMLERİ 309
15.1. DÂHİLİ EEPROM 309
15.1.1. EEPROM Kesmesi (#INT_EEPROM) 310
15.2. DÂHİLİ EEPROM OKUMA-YAZMA FONKSİYONLARI 310
15.2.1. READ_EEPROM() Fonksiyonu 311
15.2.2. WRITE_EEPROM() Fonksiyonu 311
15.3. FLASH PROGRAM BELLEĞİ OKUMA-YAZMA İŞLEMLERİ 311
15.3.1. READ_PROGRAM_EEPROM() Fonksiyonu 311
15.3.2. WRITE_PROGRAM_EEPROM() Fonksiyonu 312
15.3.3. ERASE_PROGRAM_EEPROM() Fonksiyonu 312
15.4. DATA EEPROM VE FLASH PROGRAM HAFIZASI KORUMA İŞLEMLERİ 312
15.5. DÂHİLİ EEPROM UYGULAMASI (HAFIZALI KRONOMETRE) 313
15.6. FLASH PROGRAM HAFIZASI UYGULAMASI 317
BÖLÜM 16: ANALOG KARŞILAŞTIRICI VE VOLTAJ REFERANS MODÜLÜ 321
16.1. KARŞILAŞTIRICI (COMPARATOR) MODÜLÜ 321
16.1.1. Karşılaştırıcı Kesmesi 323
16.2. SETUP_COMPARATOR() FONKSİYONU 324
16.3. KARŞILAŞTIRICI VOLTAJ REFERANS MODÜLÜ 324
16.4. SETUP_VREF() FONKSİYONU 326
16.5. KARŞILAŞTIRICI VE VOLTAJ REFERANS MODÜLÜ UYGULAMASI 326
BÖLÜM 17: PARALEL SLAVE PORT (PSP) MODÜLÜ İŞLEMLERİ 329
17.1. PARALEL SLAVE PORT (PSP) MODÜLÜ 329
17.1.1. PSP Modülü Yazma İşlemi 331
17.1.2. PSP Modülü Okuma İşlemi 332
17.1.3. Paralel Slave Port Kesmesi (#INT_PSP) 332
17.2. SETUP_PSP() FONKSİYONU 332
17.3. PSP_INPUT_FULL(), PSP_OUTPUT_FULL(), PSP_OVERFLOW() FONKSİYONLARI 333
17.4. PARALEL SLAVE PORT MODÜLÜ UYGULAMASI 333
BÖLÜM 18: RS232 SERİ İLETİŞİM İŞLEMLERİ 337
18.1. RS232 SERİ İLETİŞİM 337
18.2. MAX232 ENTEGRESİ 340
18.3. RS232 SERİ İLETİŞİM KESMELERİ 341
18.4. #USE RS232() FONKSİYONU 341
18.5. SET_UART_SPEED() FONKSİYONU 342
18.6. RS232 GİRİŞ/ÇIKIŞ FONKSİYONLARI 342
18.7. RS232 SERİ İLETİŞİM UYGULAMASI 343
BÖLÜM 19: I^2C İLE SPI SERİ İLETİŞİM VE HARİCİ BELLEK İŞLEMLERİ 347
19.1. I^2C SERİ İLETİŞİM 348
19.1.1. #USE I^2C () Fonksiyonu 351
19.1.2. I^2C_START(), I^2C_WRITE(), I^2C_READ(), I^2C_STOP(), I^2C_POLL() Fonksiyonları 351
19.1.3. CCS C Harici EEPROM Kütüphaneleri 352

19.1.4. 24C02 EEPROM Entegresi 353
19.1.5. I^2C ve SPI Kesmesi 354
19.1.6. CCS C Dosyalarını Kullanarak 24C02 Harici EEPROM Uygulaması 354
19.1.7. Kendi Harici EEPROM Fonksiyonlarımızı Yazma 358
19.1.8. Kendi Harici EEPROM Fonksiyonlarımız İle 24C02 EEPROM Uygulaması 360
19.2. SPI SERİ İLETİŞİM 362
19.2.1. SETUP_SPI(), SPI_READ(), SPI_WRITE(), SPI_DATA_IS_IN() Fonksiyonları 362
BÖLÜM 20: MOTOR UYGULAMALARI 365
20.1. DC MOTOR 365
20.1.1. L298 DC Motor Sürücü Entegresi 366
20.1.2. DC MOTOR UYGULAMASI 367
20.2. STEP (ADIM) MOTOR 370
20.2.1. ULN2003 Entegresi 372
20.2.2. Step Motor Uygulaması (ULN2003A İle) 372
20.2.3. Bilgisayar İle Step Motor Kontrolü 375
20.3. R/C SERVO MOTOR 378
20.3.1. R/C Servo Motor Uygulaması 379
BÖLÜM 21: 1-WIRE (TEK HAT) İLETİŞİM 385
21.1. 1 WIRE (TEK HAT) İLETİŞİMİ 385
21.1.1. 1-Wire İletişim İşlemleri 387
21.1.1.1. Reset Sinyali Gönderme 389
21.1.1.2. Bir Bit 1 Bilgisi Yazma 389
21.1.1.3. Bir Bit 0 Bilgisi Yazma 390
21.1.1.4. Bir Bit Okuma İşlemi 390
21.2. 1-WIRE İLETİŞİM İÇİN KENDİ FONKSİYONLARIMIZI OLUŞTURMA 390
21.2.1. 1-Wire İletişim Hızını Seçme Fonksiyonu 390
21.2.2. 1-Wire İletişim Reset Sinyali Fonksiyonu 391
21.2.3. 1-Wire İletişimde 1 Bit Bilgi Yazma Fonksiyonu 391
21.2.4. 1-Wire İletişimde 1 Bit Bilgi Okuma Fonksiyonu 392
21.2.5. 1-Wire İletişimde 1 Byte Bilgi Yazma Fonksiyonu 392
21.2.6. 1-Wire İletişimde 1 Byte Bilgi Okuma Fonksiyonu 393
21.3. DS1990A İ-BUTTON 394
21.4. 1-WIRE İLETİŞİM UYGULAMASI (DS1990A İLE) 395
21.5. CCS C 1-WIRE İLETİŞİM KÜTÜPHANE DOSYASI 398
BÖLÜM 22: SICAKLIK SENSÖRÜ, RTC, DİJİTAL POT VE SES UYGULAMALARI 401
22.1. DS1621 DİJİTAL SICAKLIK SENSÖRÜ 401
22.1.1. DS1621 Sıcaklık ve Termostat Entegresi Uygulaması 405
22.2. DS1302 RTC (REAL TIME CLOCK/CALENDAR) 409
22.2.1. CCS C Derleyicisi DS1302 Kütüphane Dosyası 410
22.2.2. DS1302 RTC Uygulaması 411
22.3. DS1868 DİJİTAL POTANSİYOMETRE ENTEGRESİ 414
22.3.1. CCS C Derleyicisi DS1868 Kütüphane Dosyası 415
22.3.2. DS1868 Dijital Potansiyometre Uygulaması 416
22.4. CCS C DERLEYİCİSİ SES DOSYASI 418
22.4.1. Ses Uygulaması 419
DİZİN 421
KAYNAKÇA 425

NOT: *Taralı olan satırlar uygulama programı olduğunu gösterir.*

1. Bölüm

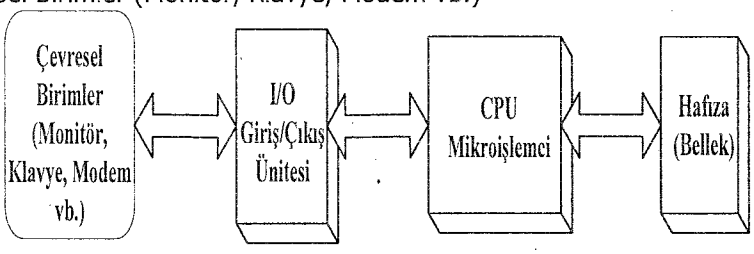
Giriş

- Mikroişlemcilere Genel Bir Bakış
- Mikrodenetleyicilere Genel Bir Bakış
- Mikrodenetleyici Programlamak İçin Gerekenler

1.1. GİRİŞ

Günümüzde mikroişlemci kelimesi pek çok insan için yabancı olmayan bir kelimedir. Elektronik ve Bilgisayar ile teknik bazda fazla ilgilenmeyen pek çok kişi bile mikroişlemcinin, bir kişisel bilgisayarda (PC) kullanılan, bir elektronik parça olduğunu bilir. Bilgisayarlardaki işlemciler, 1950'lerden günümüze çok değişik şekiller almıştır. Mikroişlemciler bilgisayara yüklenen programların tüm işlemlerini yerine getirdiği için "merkezi işlem birimi (**CPU- C**entral **P**rocessing **U**nit) olarak da adlandırılır. Genel bir bilgisayar dört ana birimden oluşur:

1- Merkezi İşlem Birimi (Central Processing Unit-CPU)
2- Hafıza - Bellek (Memory)
3- Giriş/ Çıkış(I/O) Ünitesi
4- Çevresel Birimler (Monitör, Klavye, Modem vb.)



Şekil-1.1. *Genel bir bilgisayarın bileşenleri.*

CPU, bilgisayarın değişik birimleri arasındaki veri akışı ve veri işleme görevlerini yerine getirir. Veri işlemenin çoğu, CPU içindeki aritmetik lojik birimde (ALU) gerçekleştirilir. CPU çeşitli komutları yürüterek, bilgisayar sisteminin çalışmasını yönetir. Bilgisayar hafızasında saklı olan komutlar, bilgisayarın yürüteceği programı oluşturur.

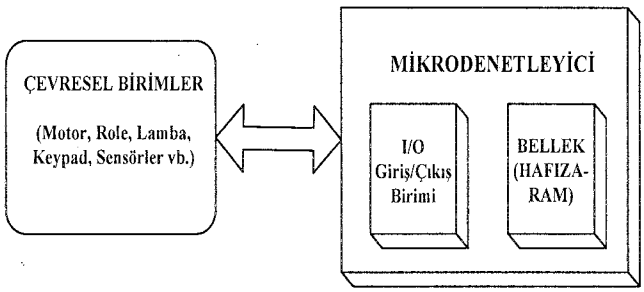
Dış dünyadan CPU' ya veri transfer eden alt birim giriş cihazıdır. Giriş verisi, bir insan, diğer bir bilgisayar veya başka bir elektronik sistem gibi çok değişik kaynaklardan gelebilir. CPU tarafından işlenen veri, hemen çıkışa gönderilebilir veya daha sonra işlenmek üzere hafızada saklanır. CPU'dan dış dünyaya veri transfer eden alt birim çıkış cihazıdır. Bu birim göstergeler, yazıcılar, diğer bilgisayarlara veya elektronik sistemlere veri transfer eden cihazları belirtmektedir.

1971 yılında, bir bilgisayarın ALU ile kontrol birimlerinin bir silikon tüm devresi üzerine, INTEL firması tarafından konulması ile mikroişlemciler dönemi başlamıştır. Basit bir teknik tanım ile mikroişlemci, yarı iletken tek bir tüm devre üzerine yerleştirilmiş CPU"ya denir. Mikroişlemciler genel amaçlı cihazlar olup birçok uygulama için uygundur. Bir mikroişlemci temel olarak CPU' nun fonksiyonlarını yerine getirdiği için, pek çok kere, CPU ile mikroişlemci eş anlamda kullanılmaktadır. Bir bilgisayarın CPU' su baskı devre kartında bulunuyorsa, bilgisayar, minibilgisayar diye adlandırılır. Bir mikroişlemci etrafına kurulu bilgisayar, mikrobilgisayar diye adlandırılır. Bir mikrobilgisayarın mikroişlemcisi, I/O ve hafıza cihazları, teknolojiye ve uygulamalara göre devamlı değişim göstermektedir. Teknolojinin gelişimi doğrultusunda, daha önce mikroişlemci tüm devresi üzerinde olmayıp dışarıda olan, pek çok giriş/ çıkış ve hafıza alt birimleri, CPU üzerine taşınmıştır.

1.2. MİKRODENETLEYİCİLERE GENEL BİR BAKIŞ

Mikrodenetleyici (Microcontroller), bir tüm devre üzerinde üretilen bilgisayara denir. Bir mikrodenetleyici tüm devresinde bulunan hafıza ve giriş/çıkış alt sistemleri (I/O), bu işlemcilerin birçok uygulama içinde, gömülü olarak doğrudan ve tek başına, mikroişlemcilere göre çok daha basit ve ucuz arabirim teknikleriyle, kontrol amaçlı olarak kullanılmalarını sağlar. Şekil-1.2'de genel olarak mikrodenetleyici sistemi verilmiştir. Günümüzde mikrodenetleyiciler; TV sistemlerinde, kameralarda, cep telefonlarında, telefon santrallerinde, otomobillerde, fotokopi cihazlarında, faks-modemlerde, oyuncaklarda ve çeşitli haberleşme cihazlarında, kontrol sistemlerinde sayılamayacak kadar pek çok alanda kullanılmaktadır.

Mikroişlemci ile bir sistem oluşturulurken Şekil-1.1'de görüldüğü gibi mikroişlemcinin yanına diğer birimleri de ilave ederek bir sistem oluşturmak gerekir. Bu ünitelerin birbirleri ile iletişimini sağlamak içinde bir data yolu (data bus) gereklidir. Ayriyeten her bir birim bir entegreden ibaret olduğundan maliyet artar ve tasarlanan baskı devre üzerinde yer işgaline sebep olur. Mikrodenetleyiciler de ise çevresel birimler hariç ihtiyaç duyulan tüm birimler tek bir yonga içindedir. Bunun bize hem maliyet, hem tasarım alanı hem de programlamada pozitif bir katkı sağlayacağı açıktır.



Şekil-1.2. *Bir Mikrodenetleyici Sisteminin Blok Şeması.*

Mikroişlemciler daha çok bilgisayar sistemleri için uygun bir elemandır. Aşırı yüksek hız gerektirmeyen, fazla belleğe ihtiyaç duyulmayan bir kontrol sistemi için mikroişlemci kullanmak hem çok pahalı hem de programlaması zor olan bir iştir. Bu nedenle ihtiyaca uygun özellikte mikrodenetleyici kullanarak tüm kontrol sistemleri kolaylıkla kontrol edilebilir. Günümüzde mikrodenetleyici üreten değişik firmalar bulunmaktadır. Intel, Microchip, Motorola, Maxim-Dallas, Parallax (Basic Stamp), Philips, Zilog gibi firmalar mikrodenetleyici üreten firmalardan bazılarıdır. Her firma kendine göre değişik özellikte ve modelde denetleyiciler üretmektedir. Her firmanın denetleyicileri tasarımlarına göre farklı komutlarla programlanmaktadır. Fakat bir mikrodenetleyici programlamasında uzman olan bir kişi başka bir mikrodenetleyiciye daha kolay uyum sağlar. Çünkü firmaların ürettiği denetleyicilerin komutları her ne kadar farklı da olsa, genel olarak denetleyici programlamasını bilen biri için bu uyum süresi fazla olmaz.

1.3. MİKRODENETLEYİCİ PROGRAMLAMAK İÇİN GEREKENLER

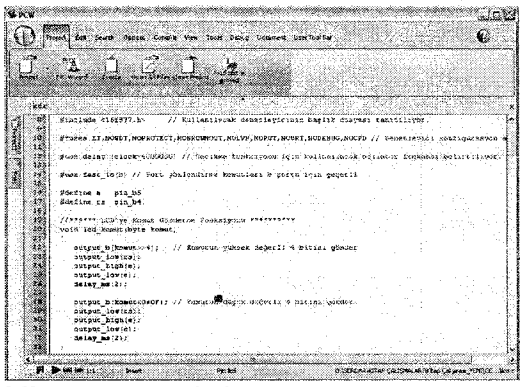
Genel olarak bir mikrodenetleyici programlanması için aşağıdaki donanım ve yazılım araçlarının olması gerekmektedir.

Bilgisayar

Kitapta anlatılan CCS C programı için kullanılacak bilgisayar IBM uyumlu bir PC olması gerekmektedir. CCS C hem Windows işletim sisteminde hem de Linux işletim sisteminde çalışabilmektedir. Kitapta anlatılan tüm işlemler Windows işletim sisteminde çalışıldığı varsayılarak anlatılmıştır.

Kullanılacak mikrodenetleyiciye uygun bir derleyici program

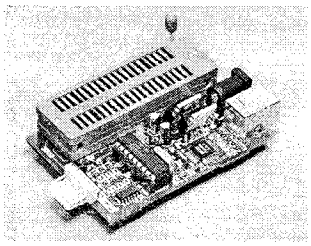
Değişik firmanın ürünü olan mikrodenetleyiciler için farklı farklı derleyici (compiler) programlar bulunmaktadır. Kullanılan denetleyiciye uygun bir derleyici program seçilmesi gerekmektedir. Kitapta anlatılan derleyici, CCS firmasının PIC denetleyiciler için yazdığı **CCS C (PCW)** derleyici programıdır.



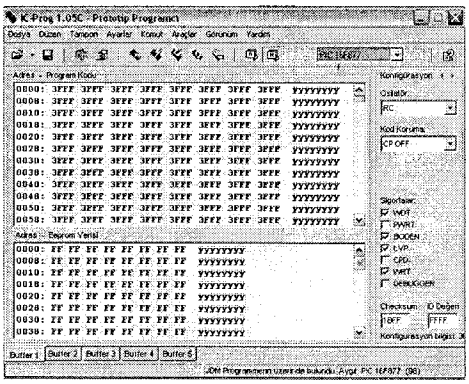
Şekil-1.3. *CCS C derleyici programının görünümü.*

Kullanılacak mikrodenetleyiciye uygun programlayıcı donanımı

Mikrodenetleyicileri programlamak için programlama uçları ile bilgisayar çıkış port'u (seri, paralel, USB) arasında belli bir bağlantı olması gerekir. Bunun için genelde programlayıcı devreler kullanılır. Bu devrelerin avantajı, aynı devre ile bir çok değişik denetleyici modelini programlayabilmesidir. Piyasa da çeşitli PIC programlayıcı devreler rahatlıkla bulunmaktadır. Aynı zamanda internet üzerinden rahatça bulabileceğiniz değişik programlayıcı devrelerinden istediğinizi de kendiniz yapabilirsiniz.



Şekil-1.4. *PIC Programlayıcı bir devre.*



Şekil-1.5. *IC-Prog programı ara yüzü.*

Kullanılacak mikrodenetleyiciye uygun programlayıcı yazılımı

Denetleyicilere yazıp derlediğiniz dosyaları yüklemek için kullandığınız **programlayıcı devresi ile uyumlu** bir yazılıma ihtiyaç vardır. IC-Prog, WinPic, Ponyprog gibi programlar ile yazılımınızı PIC denetleyicinize yükleyebilirsiniz. IC-Prog programı genelde çoğu programlayıcı devreleri desteklemektedir. Aynı zamanda Türkçe dil desteğinin de olması artı bir avantajdır.

2. Bölüm

PIC16F877 Mikrodenetleyicisi

- PIC Denetleyiciler ve PIC16F877'nin Genel Özellikleri
- Bellek Organizasyonu
- Besleme Bağlantıları
- Osilatör Konfigürasyonları
- Reset Devresi ve Çeşitleri
- Giriş/Çıkış Port'ları
- Konfigürasyon Bit'leri
- Kesmeler

2.1. PIC DENETLEYİCİLER VE PIC16F877'NİN GENEL ÖZELLİKLERİ

Bu bölümde kitapta uygulamalarda kullanılan PIC16F877 mikrodenetleyicisinin temel donanımı hakkında bilgi verilmiştir. Donanım özelliklerinin ayrıntıları ve diğer donanım bileşenleri ise uygulama örneklerinde programlama yapılırken incelenmiştir. PIC ismi Microchip firmasının ürettiği mikrodenetleyicilere verdiği isimdir. Microchip firması çeşitli özellikte ve tasarımda onlarca değişik tip denetleyici üretmektedir. Üretilen bu mikrodenetleyiciler, denetleyicinin kelime boyuna (Word Lenght) göre değişik aile (family) isimlerine ayrılmıştır. 8 bit'lik PIC mikrodenetleyici çeşitleri aşağıdaki gibi başlıklarda piyasada bulunmaktadır.

PIC10FXX - Maksimum uygulama frekansı 8 Mhz.
PIC12CXXX/PIC12FXXX - Maksimum uygulama frekansı 20 Mhz.
PIC14000 - Maksimum uygulama frekansı 20 Mhz.
PIC16C5XX - Maksimum uygulama frekansı 20 Mhz.
PIC16C7XX - Maksimum uygulama frekansı 20 Mhz.
PIC16F62X - Maksimum uygulama frekansı 20 Mhz.
PIC16F8X - Maksimum uygulama frekansı 20 Mhz.

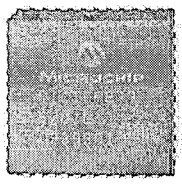
PIC16F8XX - Maksimum uygulama frekansı 20 Mhz.
PIC16C9XX - Maksimum uygulama frekansı 20 Mhz.
PIC17CXXX - Maksimum uygulama frekansı 33 Mhz.
PIC18FXXX - Maksimum uygulama frekansı 40 Mhz.

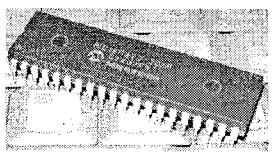
8 bit'lik PIC denetleyicilerinin yanında Microchip firmasının aynı zamanda 16 bit'lik denetleyicileri ve 16 bit'lik DSC (**D**igital **S**ignal **C**ontroller) ürünleri mevcuttur. Bunlarda aşağıda verilen başlıklarda toplanmıştır.

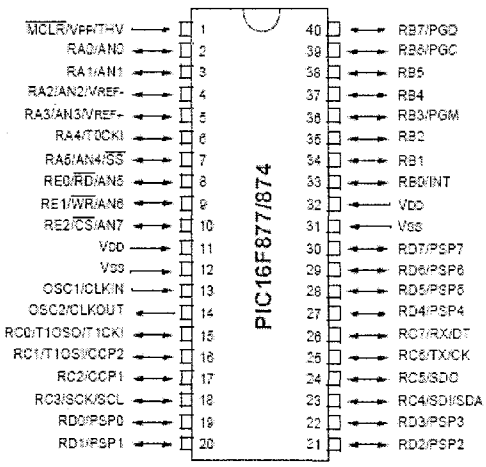
PIC24F - Maksimum uygulama frekansı 16 Mhz.
PIC24H - Maksimum uygulama frekansı 40 Mhz.
dsPIC30 DSC - Maksimum uygulama frekansı 20, 30 Mhz.
dsPIC33 DSC - Maksimum uygulama frekansı 20, 30, 40 Mhz.

Yukarıda verilen tüm PIC aileleri kendilerine has belli başlı özelliklere sahiptir. Bu aileler denetleyici kelime boyunun kaç bit olduğuna, çalışma hızlarına, giriş/çıkış port sayılarına, bellek tiplerine, içlerine dahil ettikleri diğer birimlere göre çeşitli modellere ayrılmaktadır. Biz bu kitapta piyasada çok rahat olarak bulunan ve içinde ihtiyaç duyulan çoğu özelliği barındıran PIC16F877 ürünü üzerinde programlarımızı yapacağız. Fakat burada önemli olan **CCS C derleyici ile C dilini kullanarak bir PIC programlamayı bilmektir**. Zaten derleyici programını ve PIC16F877 denetleyicisini programlamayı bilen birisi diğer PIC ürünlerini de rahatça programlayabilir.

PIC16F877 mikrodenetleyicisi PIC ailesinin güçlü bir ürünüdür. Şekil-2.1'de PIC16F877'nin pin diyagramı görülmektedir. Çeşitli tipte kılıflarda satılan bu denetleyicilerden biz uygulamalarımızda PDIP (**P**lastic **D**ual **I**n-line **P**ackage) kılıfa sahip olanlarını kullanacağız.

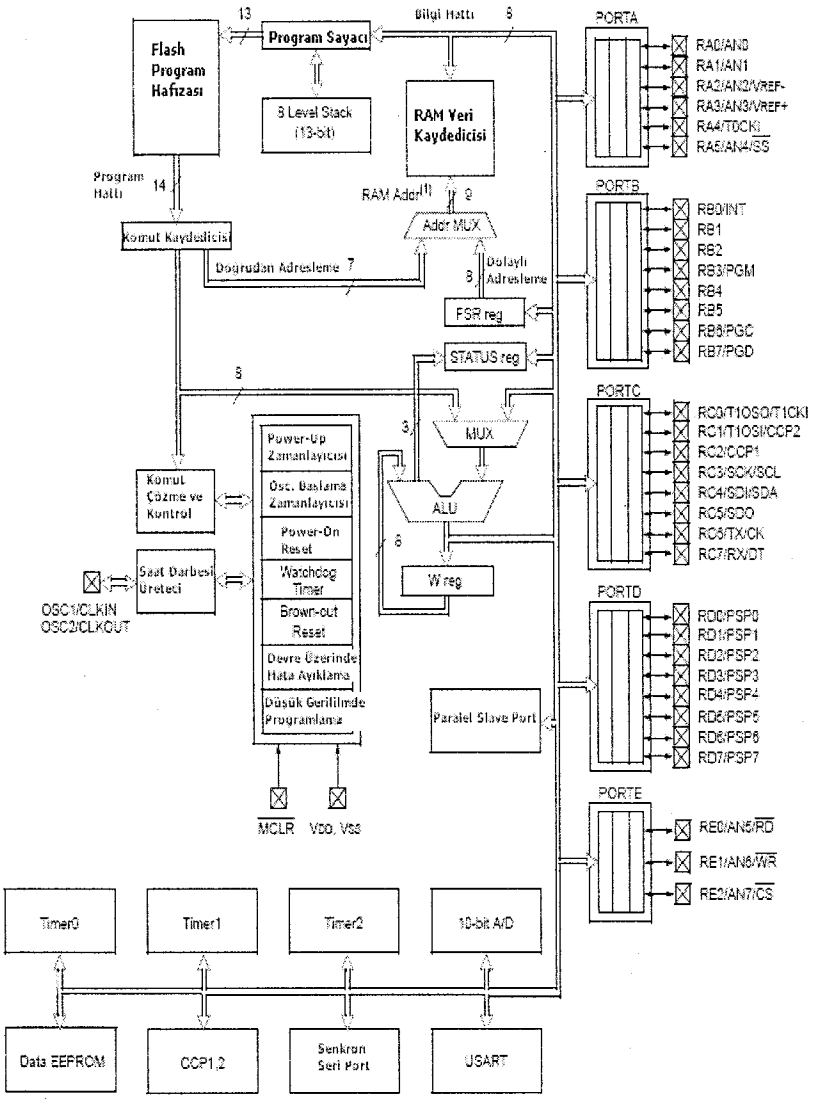






Şekil-2.1. *Çeşitli PIC16F877'nin görünümleri* ***Şekil-2.2.*** *PIC16F877 pin diyagramı*

PIC16F877'nin genel özelliklerine bakacak olursak çoğu uygulama için yeterli bir donanıma sahip olduğunu görürüz. Şekil-2.3'teki PIC16F877'nin blok diyagramı dikkatlice incelendiğinde denetleyicinin donanım mimarisini anlamak daha da kolaylaşacaktır. PIC16F877 denetleyicisi ile çoğunlukla aynı donanıma sahip olan PIC16F876 denetleyicisinin genel özelliklerini karşılaştırmalı olarak Tablo-2.1'de rahatça görebilirsiniz. Buradan da anlaşılacağı üzere aralarında fazla bir fark olmasa bile PIC16F877, PIC16F876'dan biraz daha üstündür fakat PIC16F876 ekonomik olarak daha ucuzdur.

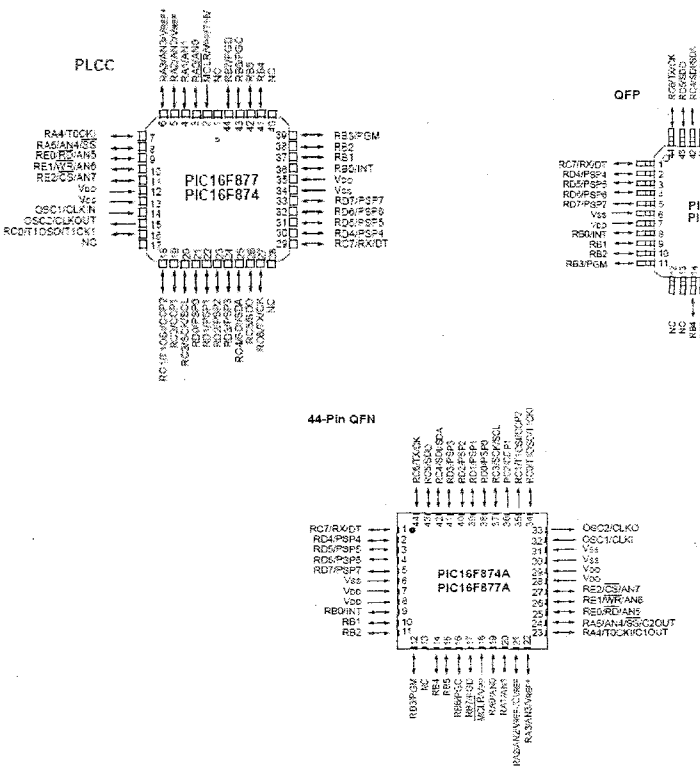


Şekil-2.3. *PIC16F877'nin blok diyagramı.*

ÖZELLİKLER	PIC16F876	PIC16F877
Uygulama Frekansı (Operating Frequency)	DC - 20 Mhz	DC - 20 Mhz
Resetler ve Gecikme (Reset and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Hafızası (Flash Program Memory) (14 bit kelime boyu)	8K	8K
Bilgi Hafızası (Data Memory) (Byte)	368	368
EEPROM Hafızası (EEPROM Data Memory)	256	256
Kesme Sayısı (Interrupts)	13	14
Giriş/Çıkış Port Sayıları (I/O Ports)	PORT A,B,C	PORT A,B,C,D,E
Zamanlayıcı Sayıları (Timers)	3	3
Yakala/Karşılaştır/PWM Modülü (Capture/Compare/PWM Modules)	2	2
Seri İletişimler (Serial Communications)	MSSP, USART	MSSP, USART
Paralal İletişimler (Parallel Communications)	-	PSP
10 Bit Analog Dijital Çevirici (Analog to Digital Module)	5 Kanal	8 Kanal
Komut Sayıları (Instruction Set)	35 Komut	35 Komut

Tablo-2.1. *PIC16F876 ve PIC16F877'nin özellikleri.*

40 pin PDIP kılıfı dışında PIC16F877 ve PIC16F877A denetleyicileri, 44 pin QFP (**Q**uad **F**lat **P**ackage), 44 pin MQFP (**M**etric **Q**uad **F**lat **P**ackage), 44 pin TQFP (**T**hin **Q**uad **F**lat **P**ackage), 44 pin PLCC (**P**lastic **L**eaded **C**hip **C**arrier), QFN (**Q**uad **F**lat **P**ackage **N**o **L**eads) kılıflarında da üretilmektedir. QFP ve QFN kılıfları birbirine benzer olmakla beraber aralarında tek bir fark vardır. QFP kılıflarında pin uçları dışarıdayken, QFN kılıflarında pin uçları dışarıya çıkartılmamıştır. Uçlar kılıfın altında metal yüzey şeklindedir.



Şekil-2.4. *Değişik PIC16F877 ve PIC16F877A kılıfları pin diyagramları.*

2.2. BELLEK ORGANİZASYONU

Bu kısımda PIC16F877'nin bellek organizasyonu hakkında genel bilgi verilmiştir. CCS C ile PIC programlarken bellek işlemlerini derleyici kendisi yaptığından dolayı bu kısımda ayrıntılı bilgi verilmekten kaçınılmıştır. Ayrıntılı bilgi için denetleyicinin datasheet'ine (veri sayfası) bakabilirsiniz. PIC16F877 belleği üç kısımdan oluşmaktadır. Bunlar **Program belleği yani Flash bellek**, **Data (veri)** belleği ve **EEPROM data** belleğidir.

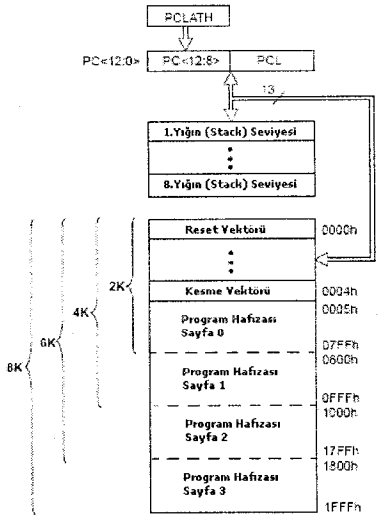
2.2.1. Program Belleği (Flash Bellek)

Program belleği her biri 2K'lık 4 sayfadan oluşur. Her sayfa sırayla,

0h - 7FFh, 800h - FFFh, 1000h - 17FFh, 1800h - 1FFFh adreslerini içerir.

Bu adreslere erişim ise **program sayacı** (program counter) ile sağlanır. Program sayacı 13 bit'liktir ve 8Kx14 değerinde adresleme kapasitesine sahiptir. Burada 14 sayısı PIC denetleyicilerindeki komut uzunluğudur. Dolayısıyla 8x1Kb = 8x1024bit =

8192 kadar program belleği adreslenebilmektedir. Şekil-2.5'te verilen program belleği haritası incelendiğinde konu daha iyi anlaşılacaktır. Şekilden görüldüğü gibi PIC denetleyicilerde **Reset vektörünün** adresi 0000h, **kesme vektörünün** adresi ise 0004h'tır.



Şekil-2.5. *Program hafızası haritası.*

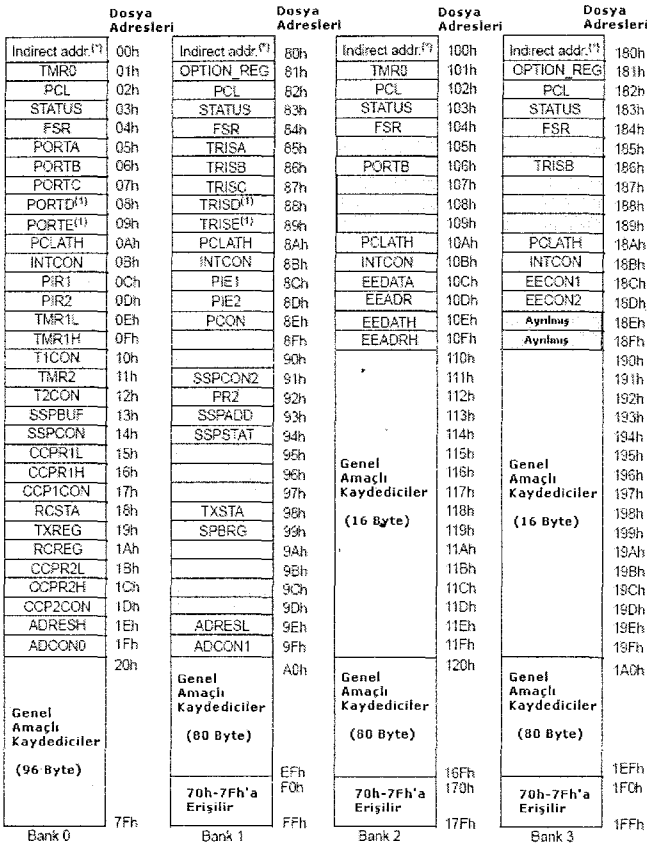
2.2.2. Veri Belleği (RAM Bellek)

Veri belleği 4 adet bank'tan oluşan **genel amaçlı kaydedicilerden** ve **özel amaçlı kaydedicilerden** -SFR- (**S**pecial **F**unction **R**egister) oluşmaktadır. Genel amaçlı kaydediciler, programcı tarafından program geliştirirken kullanılan yazılım amaçlı kaydedicilerdir. Örneğin bir program yazarken tanımlanan bir değişken programcı tarafından belirlenen genel amaçlı veri belleğinde istenen bir adrese veya derleyici program tarafından otomatik olarak belirlenen bir adrese kaydedilir. Bu kayıtlar geçicidir. Enerji kesildiğinde veya programda yapılan değişiklikle adreslerdeki bilgiler silinir veya değişir. Özel amaçlı kaydediciler ise denetleyici donanımını kontrol etmek için kullanılan kaydedicilerdir. Örneğin CCPR1L özel amaçlı kaydedicisine CCP1 birimini kontrol etmek için gerekli 16 bit'lik kontrol bilgisinin düşük seviyeli 8 bit'i yazılır. Bunun gibi özel amaçlı kaydediciler ile denetleyici donanımı kontrol edilmektedir. Şekil-2.6'da RAM veri belleği adresleri anlaşılır olarak görülebilmektedir. Şekilden de görüleceği üzere 4 adet bank'tan oluşan RAM belleğinin her bir bank'ında bulunan adresler dosya adresi (File Address) olarak isimlendirilmiştir.

Veri belleğindeki her bank 128 Byte (7Fh) kapasitesindedir. Yani her bank'ta 8 bit'lik 128 adet kaydedici bulunmaktadır. Dolayısıyla toplamda 4x128=512 adet kaydedici mevcuttur. Şekil incelendiğinde özel amaçlı kaydedicilerden bazılarının

birden fazla bank'ta bulunduğu görülmektedir. Bu şu anlama gelmektedir. Birden fazla bank'ta yer alan bir kaydedicinin değeri her hangi bir bank'ta değiştirildiğinde bu değişiklik diğer bank'larda da etkili olmaktadır. Bir bakıma bir çeşit aynalama işlemi oluşmaktadır. Bu aynalama mantığının nedeni kaydedicilere hızlı erişmek ve programlamada daha az komut kullanmaktır. Çünkü 4 bank'ın hepsinde olan bir kaydediciye o anda bulunan bank'tan hemen erişilebilir. Diğer türlü örneğin 1. bank'ta iken 3.bank'ta bulunan bir kaydediciye erişmek için 1. bank'tan 3. bank'a geçmek gereklidir. Bu bize hem zaman kaybı verir hem de daha fazla komut kullanmamıza neden olur. Bank'ların son 16 byte'ları da birbiri ile aynalanmıştır. Bir bank'ın son 16 byte'nın her hangi birine yazılmış bir veri diğer bank'ların son 16 byte'lık kısımda aynı yerde görünür. Bu nedenle bu kısma gölgelenmiş (shadowed) bellek adı da verilmektedir.

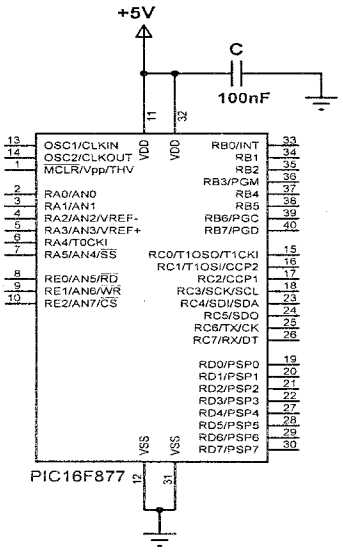
Veri belleği haritasında bazı adresler gri olarak boyanmıştır. Bunlar kullanılmayacak olan kaydedicilerdir. Bu adresler yazılım kısmında kullanılmamalıdır. Bu adresteki kaydedicilerin değeri her zaman "0" değerindedir.



Şekil-2.6. *RAM veri belleği haritası.*

2.3. BESLEME UÇLARI VE BAĞLANTILARI

PIC16F877 için besleme uçları; V_{SS} (Negatif besleme), 12 veya 31 nolu pin'lerden birine, V_{DD} (Pozitif besleme) 11 veya 32 nolu pin'lerden birine uygulanır. Besleme gerilimleri değişik durumlara göre 2.0V ile 5.5V arasında olabilir. Genelde laboratuar çalışmalarında 5V idealdir. Devreye enerji verildiği anda meydana gelebilecek gerilim dalgalanmaları nedeniyle istenmeyen arızaları önlemek amacıyla V_{DD} ile V_{SS} arasına 0.1 F'lık (100 nF) bir dekuplaj kondansatörü bağlamak gerekir. PIC16F877 4 Mhz çalışma frekansında 5V beslemede 2 mA civarında, 3V beslemede 32 Khz çalışma frekansında 20 µA civarında akım çekerler. Bekleme anında ise 1 µA civarında akım çekerler. Ayrıca denetleyicinin sink ve source akımı 25 mA'dir.



Şekil-2.5. *PIC16F877 besleme uçları ve bağlantıları.*

2.4. OSİLATÖR KONFİGÜRASYONLARI

PIC hafızasındaki komutları işlemek için bir sinyale ihtiyaç duyar. Bu sinyale clock sinyali yani saat sinyali denir. PIC bu clock sinyalini osilatör uçlarına bağlanan osilatör devresinden alır. PIC16F877'nin osilatör uçları 13 (OSC1) ve 14 (OSC2) dür. **PIC girişine bağlanan osilatör frekansını 4'e bölerek kullanılır.** Yani girişine 20 Mhz'lik bir osilatör bağlanmışsa PIC bunu 4'e böler ve sonuçta 5 Mhz'lik bir uygulama frekansı ile çalışır. Bu da;

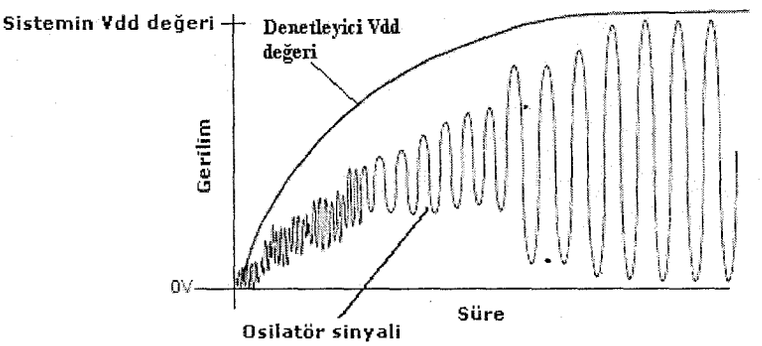
$T \quad \frac{1}{f} \quad \frac{1}{5Mhz} \quad \frac{1}{5.10^6 hz} \quad 0.0000002sn \quad 0.2 \ sn$'de bir komut işleneceği anlamına gelir.

PIC denetleyicilere bağlanabilen osilatör tipleri şunlardır.

LP - Düşük Güçlü Kristal Osilatör (Low Power)
XT - Kristal/Resonatör Osilatör (Crystal/Resonator)
HS - Yüksek hızlı kristal/Resonatör Osilatör (High Speed)
RC - Direnç-Kondansatör Osilatör (Resistor/Capacitor)

Denetleyiciye bağlanan değişik osilatör türlerinin hepsinde istenen osilatör frekansı, besleme geriliminin (Vdd) normal değere gelmesi anında elde edilir. Besleme gerilimi de hemen istenen değere ulaşamaz. Sonuçta devreye enerji verildiğinde osilatör frekansının istenen değerine ulaşmasını etkileyen faktörler şunlardır.

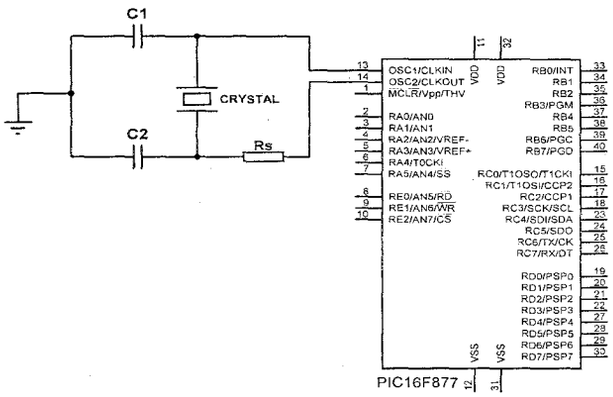
Kristal / Rezanatör frekansı
Kullanılan kapasitör değerleri (C1, C2)
Sistem sıcaklığı
Seri direnç değeri (Rs)
Kristal kalitesi
Osilatör devresi baskı devresi çizim şekli
Sistem gürültüsü
Denetleyicinin Vdd gerilimine ulaşma süresi



Şekil-2.6. *Zamana bağlı osilatör sinyal grafiği.*

2.4.1. Kristal Osilatörler

Tasarlanacak devreye göre istenen osilatör tipi seçilebilir. Şekil-2.7'de HS, XT ve LP osilatör tipleri için bağlantı şeması görülmektedir. Kristal ile bağlanacak kondansatörlerin değerleri kullanılacak kristal frekansına göre değişmektedir. Tablo-2.2'de kristal frekansına göre bağlanması gereken kondansatör değerleri verilmiştir. Kristal kesilmelerini önlemek için R_S direncinin kullanılması gerekebilir.



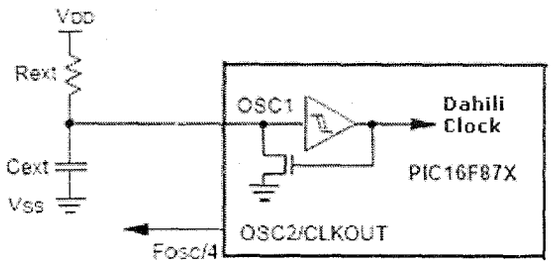
Şekil-2.7. *HS, XT ve LP Osilatörleri için bağlantı şekli.*

Osilatör Tipi	Kristal Frekansı	C1	C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 Mhz	15 pF	15 pF
	4 Mhz	15 pF	15 pF
HS	4 Mhz	15 pF	15 pF
	8 Mhz	15-33 pF	15-33 pF
	20 Mhz	15-33 pF	15-33 pF

Tablo-2.2. *Kristal frekansına göre bağlanması gereken kondansatör değerleri.*

2.4.2. R/C Osilatörü

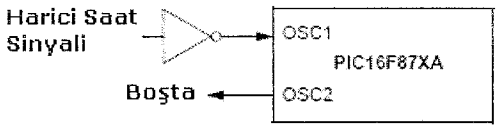
R/C osilatörler kristal osilatöre göre frekans değerlerinde çok daha fazla sapma gösterirler. Çok hassas değillerdir. Bu nedenle hassasiyet gerektiren uygulamalarda kullanılmazlar. Şekil-2.8'de R/C osilatör bağlantısı görülmektedir.



Şekil-2.8. *R/C osilatör bağlantı şeması.*

2.4.3. Harici Osilatör

PIC denetleyicileri yukarıda anlatılan osilatör bağlantılarının yanında bir de harici (dışarıdan - External) olarak osilatör frekansıyla beslenebilir. Harici bir osilatör kullanıldığında sadece OSC1 pin'i kullanılır, OSC2 pin'i kullanılmaz. Harici osilatör frekansına göre programda LP, XT veya HS modlarından biri yine belirtilmelidir. Yalnız harici osilatör uygulamalarında LP, XT veya HS modlarının maksimum frekans değerleri farklıdır. Tablo-2.3'te maksimum çalışma frekansları verilmiştir.



Şekil-2.9. *Harici osilatör bağlantısı.*

Osilatör Modu	Maksimum Sinyal Frekansı
LP	200 Khz
XT	4 Mhz
R/C	4 Mhz
HS	Max. 20 Mhz

Tablo-2.3. *Harici osilatör bağlantısında osilatör modları maksimum frekans değerleri.*

2.5. SIFIRLAMA (RESET) DEVRESİ VE ÇEŞİTLERİ

PIC16F877'in içinde aşağıda verilen çeşitlerde sıfırlama (reset) durumları mevcuttur. Bu durumlardan bazıları denetleyici tarafından kararlı bir çalışma sağlanması amacıyla otomatik olarak kullanılır. Sıfırlama işleminde tüm kaydedicilerin içerikleri sıfırlanmaz (veya başlangıç konumuna gelmez). Bu nedenle sıfırlama anında bazı belli kaydedicilerin içeriği sıfırlanırken bazı kaydedicilerin içeriği ise rasgele bir değer alacaktır.

Besleme İlk Verildiği Anda Sıfırlama - Power On Reset (**POR**)
Güç Zamanlayıcılı Sıfırlama - Power Up Timer (**PWRT**)
Osilatör Başlama Zamanlayıcısı - Oscillator Start-up Timer (**OST**)
Gerilim Düşmesi Sıfırlama - Brown Out Reset (**BOR**)
Normal Çalışmada $\overline{\text{MCLR}}$ Sıfırlama
Uyku (Sleep) Modunda $\overline{\text{MCLR}}$ Sıfırlama
Normal çalışmada **WDT** (Watchdog Timer) Sıfırlama
Uyku Modundan Çıkışta **WDT** Sıfırlama

2.5.1. Besleme İlk Verildiği Anda Sıfırlama - Power On Reset (POR)

PIC16F876 ve PIC16F877'in besleme uçlarına gerilim uygulandığında PIC'e yüklenen programın başlangıç adresinden itibaren çalışmasını sağlayan bir reset devresi vardır. Bu reset devresi PIC içerisindedir ve "Power-on-reset" olarak adlandırılır. Denetleyicinin pozitif besleme geriliminde (Vdd), 1.2V ile 1.7V arasında bir gerilim yükselmesi saptandığında POR sıfırlama sinyali meydana gelir.

2.5.2. Güç Zamanlayıcılı Sıfırlama - Power Up Timer (PWRT)

Bu resetleme durumu seçildiğinde PIC denetleyici yaklaşık 72 ms süre boyunca (T_{PWRT}) reset konumunda kalır. Süre sonunda denetleyici hafızasına yüklü programı çalıştırmaya başlar. Denetleyici besleme gerilimi ilk anda hemen normal değerinde değildir. Belli bir süre sonra güç kaynağı istenen Vdd değerine ulaşır. İşte bu durumda denetleyicinin kararsız durumlar üretmesini önlemek için PWRT resetleme durumu seçilebilir. T_{PWRT} süresi değişik denetleyici versiyonlarında değişiktir. Kullandığınız denetleyicinin veri sayfalarında (data sheet) bu süre değerleri bulunmaktadır.

2.5.3. Osilatör Başlama Zamanlayıcısı - Oscillator Start-up Timer (OST)

Osilatör başlama zamanlayıcısı 1024 adet çevrimlik bir gecikme zamanı süresince denetleyiciyi sıfırlama konumunda tutar. Bu süre kristal ve rezonatör osilatörlerin osilasyona başlama ve karalı duruma gelmesi için ihtiyaç duyulan süreyi sağlar. OST zamanlayıcısı XT, LP ve HS osilatör bağlantıları için geçerlidir.

2.5.4. Gerilim Düşmesi Sıfırlama - Brown Out Reset (BOR)

BOR devresini PIC'i programlarken konfigürasyon bit'leri ile açıp kapatabiliriz. Eğer V_{DD} gerilimi V_{BOR} gerilim değerinden (yaklaşık 4V) aşağı düşerse ve bu T_{BOR} süresi (yaklaşık 100 µsn) kadar devam ederse BOR aktif hale gelir ve PIC'i sıfırlar. V_{DD} değeri V_{BOR} değerine gelinceye kadar PIC sıfırlama konumunda kalmaya devam eder.

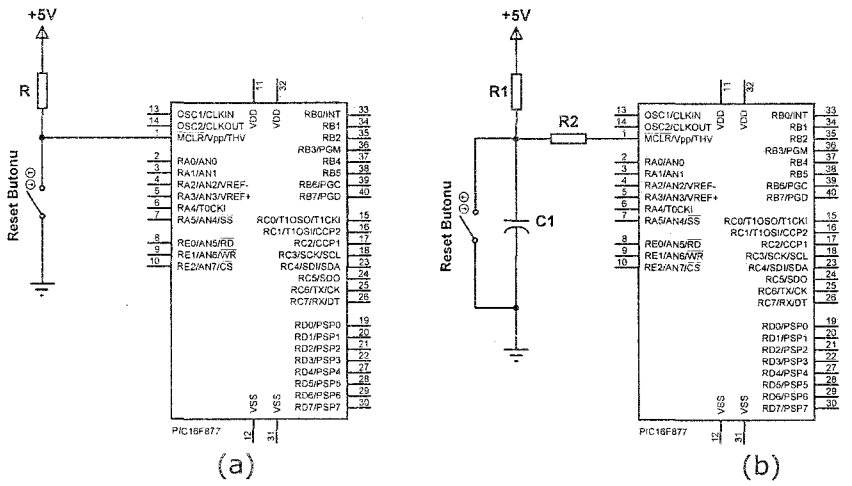
2.5.5. Watch Dog Timer (WDT) Sıfırlaması

WDT denetleyici içinde gömülü bulunan ve ek olarak dışarıdan hiçbir elemanın bağlantısına gerek duymayan bir R/C osilatördür. WDT'yi açık (enable) veya kapalı (disable) yapmak için programlamada konfigürasyon bit'lerinde bunun belirtilmesi gerekir. Konfigürasyon bit'leri ile seçilen WDT açık veya kapalı durumu yazılım aşamasında program komutlarıyla değiştirilemez. Yani WDT konfigürasyon bit'lerinde açık olarak belirtilmişse, denetleyiciye yüklenen program kodları WDT

kapatılamaz, sadece WDT'ın içeriği sıfırlanabilir. Normal çalışma sırasında WDT belli bir süre sonunda oluşturduğu sinyal ile denetleyiciyi sıfırlar. Eğer denetleyici uyku (sleep) modunda iken WDT taşma sinyali geldiğinde denetleyici uyku modundan çıkar ve normal çalışmaya programda kaldığı yerden devam eder. WDT'ın taşma sinyali üretme süresi yaklaşık 18 ms'dir. Bu değer ortam sıcaklığı, kullanılan denetleyici çeşidi ve Vdd değerlerine göre değişir. Eğer WDT'ın zaman aşımı süresi artırılmak isteniyorsa, OPTION kaydedicinde bölme oranı değeri (prescaler) değiştirilir ve bu değerin WDT için geçerli olmasını sağlayan 3. bit'i "1" yapılır.

2.5.6. $\overline{\text{MCLR}}$ Sıfırlaması

$\overline{\text{MCLR}}$ sıfırlama, dışarıdan sinyal uygulanarak mikrodenetleyiciyi sıfırlamaya yarar. $\overline{\text{MCLR}}$ ucu lojik-0'da aktiftir. Normal çalışmada denetleyicinin hep sıfırlama yapmaması için bu uç çeşitli sıfırlama devrelerine bağlanmalıdır. Böylece bir buton üzerinden denetleyicinin kendini resetlemesi sağlanır. Denetleyici $\overline{\text{MCLR}}$ ucu için bir filtre devresi barındırmaktadır. Bu sayede $\overline{\text{MCLR}}$ ucunda meydana gelen istenmeyen küçük parazit sinyallerinin kararsız bir durum oluşturma ihtimali önlenmiş olmaktadır. Denetleyici beslemesi için kullanılan güç kaynağı hız bir şekilde Vdd gerilimini sağlıyorsa Şekil-2.10-a'daki sıfırlama devresi kullanılabilir. Eğer kullanılan güç kaynağı istenen gerilim değerine yavaş ulaşıyorsa Şekil-2.10-b'deki sıfırlama devresini kullanmak daha iyi sonuç verecektir. Bu devre aynı zamanda parazitleri de önlemeye yardımcı olur. **Devrelerde R ve R1 dirençleri 4.7K, C1 kondansatörün ise 0.1 µF (100 nF) olması uygundur. R2 direnci ise 40 KΩ'dan daha küçük olmalıdır.**



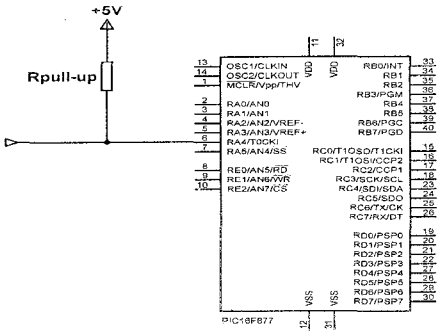
Şekil-2.10. *Hızlı (a) ve Yavaş (b) güç kaynakları için* $\overline{\text{MCLR}}$ *sıfırlama devreleri.*

2.6. GİRİŞ / ÇIKIŞ (I/O) PORT'LARI

PIC16F877 denetleyicisinde A, B, C, D, E giriş/çıkış port'ları mevcuttur. Bu port'lar isteğe göre giriş veya çıkış olarak kullanılabilir. Hangi port'un hangi bit'lerinin giriş hangi bit'lerinin çıkış olarak kullanılacağı programın başında belirtilmelidir. Giriş ve çıkış olarak belirtilen bu port'ların birden çok görevi vardır. Hangi amaçla kullanılmak isteniyorsa program ona göre yazılır. Örneğin bir uç normal dijital giriş/çıkış işlemleri için kullanılabildiği gibi analog giriş içinde kullanılabilir. Uygulama örneklerinde bu konu daha iyi anlaşılabilir. Port'ların sahip olduğu pin'ler birden fazla görev için kullanılmak üzere tasarlanmıştır. Bu nedenle programlama aşamasında hangi pin'in hangi görevde kullanılacağı bilinmeli ve işlemler ona göre yapılmalı. Örneğin A port'u hem dijital giriş/çıkış için kullanılırken, aynı zamanda da analog giriş için kullanılabilir. Bu nedenle bu uçlar dijital giriş/çıkış olarak kullanılacağında analog birim devre dışı bırakılmalıdır.

2.6.1. PORT A

A port'u hem giriş/çıkış hem de analog giriş uçları görevlerini yapan 6 bit'lik bir port'tur. Tablo-2.4'te A port'unu pin'lerinin görevleri verilmiştir. A port'un bulunan RA4 pin'i diğer PortA pin'lerinden farklıdır. RA4 pin'i açık kolektör özelliğine sahiptir. Bu nedenle bu pin'in çıkış olarak kullanılması için mutlaka pull-up direnci ile pozitif beslemeye bağlanması gerekir. Pull-up direnci 5 KΩ ile 10 KΩ arasında olabilir.



Şekil-2.11. *RA4 pin'inin çıkış olarak kullanılması.*

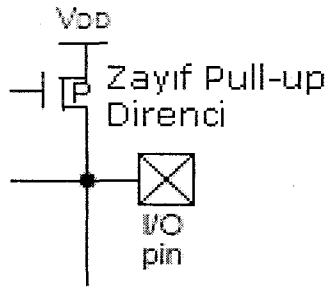
Pin Adı	Pin No	Buffer Tipi	Pin fonksiyonları
RA0/AN0	0	TTL	Dijital I/O veya Analog giriş
RA1/AN1	1	TTL	Dijital I/O veya Analog giriş
RA2/AN2	2	TTL	Dijital I/O veya Analog giriş
RA3/AN3/Vref	3	TTL	Dijital I/O veya Analog giriş vaya Vref
RA4/TOCKI	4	ST	Dijital I/O veya Timer0 dış saat sinyal girişi
RA5/SS/AN4	5	TTL	Dijital I/O veya Analog giriş veya senkron seri iletişim slave seçme girişi

TTL = Transistor Transistor Logic ST = Schmitt Trigger

Tablo-2.4. *PORTA pin açıklamaları.*

2.6.2. PORT B

PortB giriş/çıkış olarak kullanılabilen 8 bit'lik bir port'tur. B port'u denetleyici içinde dahili olarak pull-up yapılmış gibidir. Şekil-2.12'de görüldüğü gibi pin girişi FET'in kanal (drain) ile kaynak (source) arasında gösterdiği yüksek direnç ile beslemeye bağlanmış, dolayısıyla pull-up bağlantı sağlanmıştır. FET ile sağlanan bu pull-up direncine zayıf pull-up direnci denmektedir. PortB'nin bu özelliği ile bu port giriş olarak kullanıldığında girişte sinyal yokken giriş lojik-1 olarak algılanır. Bu nedenle ek dirence gerek olmadan buton girişi uygulamalarında tercih edilir. Dahili pull-up olayı programla aktif veya pasif yapılabilir.



Şekil-2.12. *PortB dahili pull-up durumu*

Pin Adı	Pin No	Buffer Tipi	Pin fonksiyonları
RB0/INT	0	TTL/ST	Dijital I/O veya Harici kesme
RB1	1	TTL	Dijital I/O
RB2	2	TTL	Dijital I/O
RB3/PGM	3	TTL	Dijital I/O veya LVP (düşük gerilimle programlama) modunda programlama girişi
RB4	4	ST	Dijital I/O
RB5	5	TTL	Dijital I/O
RB6/PGC	6	TTL/ST	Dijital I/O veya seri programlama clock pin'i
RB7/PGD	7	TTL/ST	Dijital I/O veya seri programlama data pin'i

Tablo-2.5. *PortB pin açıklamaları*

2.6.3. PORT C, PORT D ve PORT E

C port'u da 8 bit'lik dijital giriş/çıkış port'udur. C port'u diğer port'lara nazaran daha fazla fonksiyonu içinde barındırır. Tablo-2.6'ta C port'u pin'lerinin sahip olduğu tüm fonksiyonlar görülmektedir. D port'u da C port'u gibi 8 bit'lik dijital giriş/çıkış port'udur. E port'u ise 3 bit'lik dijital giriş/çıkış port'udur. Tablo-2.7'de D port'u, Tablo-2.8'de de E port'u pin açıklamaları görülmektedir.

Pin Adı	Pin No	Buffer Tipi	Pin fonksiyonları
RC0/T1OSO/T1CKI	0	ST	Dijital I/O veya TMR1 osilatör çıkışı veya TMR1 clock girişi
RC0/T1OSI/CCP2	1	ST	Dijital I/O veya TMR1 osilatör girişi veya CCP2 çıkışı
RC2/CCP1	2	ST	Dijital I/O veya CCP1 çıkışı
RC3/SCK/SCL	3	ST	Dijital I/O veya SPI ve I2C modunda senkron seri clock girişi
RC4/SDI/SDA	4	ST	Dijital I/O veya SPI modunda data girişi veya I2C modunda I/O ucu
RC5/SDO	5	ST	Dijital I/O veya senkron seri iletişimde data çıkışı
RC6/TX/CK	6	ST	Dijital I/O veya USART asenkron seri iletişimde data gönderme pin'i veya USART asenkron seri iletişimdeclock pin'i
RC7/RX/DT	7	ST	Dijital I/O veya USART asenkron seri iletişimde data girişi veya USART senkron seri iletişimde data pin'i

Tablo-2.6. *PortC pin açıklamaları.*

Pin Adı	Pin No	Buffer Tipi	Pin fonksiyonları
RD0/PSP0	0	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit0
RD1/PSP1	1	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit1
RD2/PSP2	2	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit2
RD3/PSP3	3	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit3
RD4/PSP4	4	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit4
RD5/PSP5	5	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit5
RD6/PSP6	6	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit6
RD7/PSP7	7	ST/TTL*	Dijital I/O veya Paralel Slave Port'u Bit7

* Dijital I/O seçildiğinde ST Buffer tipi, PSP seçildiğinde TTL buffer tipi aktiftir.

Tablo-2.7. *PortD pin açıklamaları.*

Pin Adı	Pin No	Buffer Tipi	Pin fonksiyonları
RE0/RD/AN5	0	ST/TTL*	Dijital I/O veya Paralel Slave Port'u modunda RD kontrol bit'i girişi veya analog giriş
RE1/WR/AN6	1	ST/TTL*	Dijital I/O veya Paralel Slave Port'u modunda WR kontrol bit'i girişi veya analog giriş
RE2/CS/AN7	2	ST/TTL*	Dijital I/O veya Paralel Slave Port'u modunda CS kontrol bit'i girişi veya analog giriş

* Dijital I/O seçildiğinde ST Buffer tipi, PSP seçildiğinde TTL buffer tipi aktiftir.

Tablo-2.8. *PortE pin açıklamaları.*

2.7. KONFİGÜRASYON BİT'LERİ

Konfigürasyon bit'leri PIC'e gerilim uygulandığında PIC'in hem donanımsal hem de yazılımsal olarak başlangıç ayarlarını bildiren bilgileri içerir. PIC'in hangi osilatör türünde çalıştırılacağı, kullanılmak istenen sıfırlama devrelerinin aktif veya pasif yapılması, denetleyiciye yüklenen program kodlarının korumaya açık veya kapalı olması gibi durumların belirlendiği bit'lerdir. Konfigürasyon bit'i kaydedicisi 14 bit'liktir ve bu kaydediciye bilgiler ya program aracılığıyla yada programı denetleyiciye yüklerken program yükleme yazılımıyla yüklenir. Program yükleme yazılımları ile PIC'e yüklenen programda belirtilen konfigürasyon bit'lerini değiştirebilirsiniz. CCS C derleyicisi anlatılırken konfigürasyon ayarları ve programa yazılması ayrıntılı olarak incelenmiştir.

2.8. KESMELER (INTERRUPTS)

Kesme işlemini değişik kaynaklardan gelen bir sinyal ile çalışan programın kesilmesi ve kesme fonksiyonuna gitmesi olarak tanımlayabiliriz. Kesme durumu programın işlenmesinden daha önemlidir. Dolayısıyla bir kesme meydana geldiğinde program işlemekte olduğu komutları bırakarak kesme işlemini gerçekleştirir. Kesme işlemi sonunda da program kaldığı yerden çalışmaya devam eder. Kesmeler donanım kaynaklı olduğundan da programı boşa komut işlemekten kurtarır. Örneğin bir sistemde belirlenen bir butona basılınca sistemin alarm komuna geçmesi istendiğini farz edelim. Bu işlemi program ile yapmaya kalkarsak programda her zaman bu buton girişini kontrol ettirmek gereklidir. Bu hem sistemin hızını yavaşlatır hem de sağlıklı bir kontrol olmaz. Çünkü program aynı zamanda başka işlemleri de yapmaktadır. Fakat biz bu buton girişini bir dış kesme donanımına bağlarsak. Buton kontrol işini donanım kesmesi birimi yapacak ve program bu devamlı kontrol işinden kurtulacaktır.

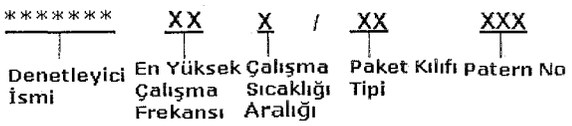
PIC16F877'de 14 adet kesme vardır. Bu kesmeler arasında da öncelik derecesi verilebilir. Yani aynı anda iki kesme meydana gelirse hangi kesmenin işleyeceği bu öncelik derecesine göre denetleyiciye bildirilir. Uygulama örneklerinde kesmeler daha iyi anlaşılabilir. Aşağıda PIC16F877 denetleyicisinde bulunan tüm kesmeler verilmiştir.

1. RB0 Harici Kesmesi.
2. RB4-RB7 Pin'lerindeki Değişiklik Kesmesi.
3. Timer0 Birimi Taşma Kesmesi.
4. Timer1 Birimi Taşma Kesmesi.
5. Timer2 Birimi Taşma Kesmesi.
6. A/D Çevrimi Yapıldığında Meydana Gelen Kesme.
7. CCP1 Modülü Kesmesi.
8. CCP2 Modülü Kesmesi.

9. Paralel Port'tan Veri Gelme Kesmesi.
10. Seri Port'tan Veri Geldiğinde Oluşan Kesme.
11. SPI veya I^2C İletişimi Sırasında Veri Gelme Kesmesi.
12. EEPROM'a Veri Yazma İşlemi Sonlandığında Oluşan Kesme.
13. RS232 Seri İletişiminde Gönderilecek Veri Tamponu Boş Olduğunda Meydana Gelen Kesme.
14. Bus Collision Kesmesi MSSP (modunda seri iletişimde hata oluştuğunda).

2.9. ÜRÜN TANIMLAMA KOD AÇILIMLARI

PIC16F87X ürünlerinin kılıflarının üstünde değişik anlamlar ifade eden kodlamalar vardır. Bu kodlar sayesinde kullandığımız PIC ürünü hakkında bazı bilgiler elde ederiz. PIC16F87X denetleyici kılıflarının üstündeki kod gruplarının anlamları Şekil-2.11'de görülmektedir. Şeklin altında da kod gruplarının aldığı değerler ve anlamları verilmiştir.



Şekil-2.11. *Ürün tanımlama kod grupları.*

Denetleyici İsmi	PIC16F87X, PIC16F87XT Besleme Gerilimi 4V - 5.5V PIC16LF87, PIC16LF87XT Besleme Gerilimi 2V - 5.5V (F = CMOS Flash, L = Düşük Güçlü)			
En Yüksek Çalışma Frekansı	04	=	4 Mhz	
	10	=	10 Mhz	
	20	=	20 Mhz	
Çalışma Sıcaklığı Aralığı	Boş	=	0 ile 70 derece	Ticari
	I	=	-40 ile +85 derece	Endüstriyel
	E	=	-40 ile +125 derece	Genişletilmiş
Paket Kılıfı	PQ	=	MQFP	
	PT	=	TQFP	
	SO	=	SOIC	
	SP	=	İnce PDIP	
	P	=	PDIP	
	L	=	PLCC	

Örneğin denetleyici kılıfı üzerinde PIC16F877 - 10I/P yazıyorsa bundan şunları anlarız. Denetleyici ismi PIC16F877, en yüksek çalışma frekansı 10 Mhz, çalışma sıcaklığı aralığı endüstriyel tip yani -40 ile +85 derece, PDIP kılıfın da ve normal besleme geriliminde (4V - 5.5V) çalışmakta olduğunu anlarız.

3. Bölüm

CCS C Derleyicisi

- CCS C Programı
- CCS C DEMO Programının Kurulması
- CCS C Derleyicisi Menüleri

3.1. CCS C PROGRAMI

CCS C derleyici programı **C**ustom **C**omputer **S**ervices Inc. firmasının bir ürünüdür. CCS firması Microchip firmasının PIC denetleyicileri için C dilinde programlanmasını sağlayan tümleşik yazılımlar geliştiren ve bu alanda deneme kartları üreten bir firmadır. CCS firması aynı zamanda başka firma ve işletmelere istenen özelliklerde yazılım ve devre kartları konusunda da donanımsal olarak mühendislik ve danışmanlık yapmaktadır.

CCS C derleyicisi üç değişik derleyiciden oluşur. Bunlar;

PCB
PCM
PCH

Kurulum yapılırken bu üç derleyicinin satın alınanlardan hepsi veya isteneni programa dahil edilebilir. **PCB derleyicisi 12 bit'lik PIC denetleyiciler için, PCM derleyicisi 14 bit'lik PIC denetleyiciler için, PCH derleyicisi ise 16 ve 18 bit'lik PIC denetleyiciler içindir. Genelde CCS C sürümlerinde bu üç derleyicide yüklenmektedir.** CCS C derleyicilerin en son sürümleri v4.xxx olanlarıdır. Sürüm(versiyon) 4 derleyicisinden önce sürüm 3 derleyicileri kullanılıyordu. Sürüm 3 ile sürüm 4 derleyicileri arasında kullanıcı arabirimi olarak çok değişiklik vardır. CCS C derleyici programı yaklaşık 325 adet PIC ürününü desteklemektedir. Bunlar şunlardır;

PIC10F200, PIC10F202, PIC10F204, PIC10F206, PIC10F220, PIC10F222

PIC12F508, PIC12C508, PIC12C508A, PIC12C509, PIC12F509, PIC12C509A, PIC12CR509A, RFPIC12C509AF, RFPIC12C509AG, PIC12F510, PIC12CE518,

PIC12CE519, PIC12HV609, PIC12F609, PIC12HV615, PIC12F615, PIC12F629, PIC12F635, PIC12C671, PIC12C672, PIC12CE673, PIC12CE674, PIC12F675, RFPIC12F675F, RFPIC12F675H, RFPIC12F675K, PIC12F683

PIC14000

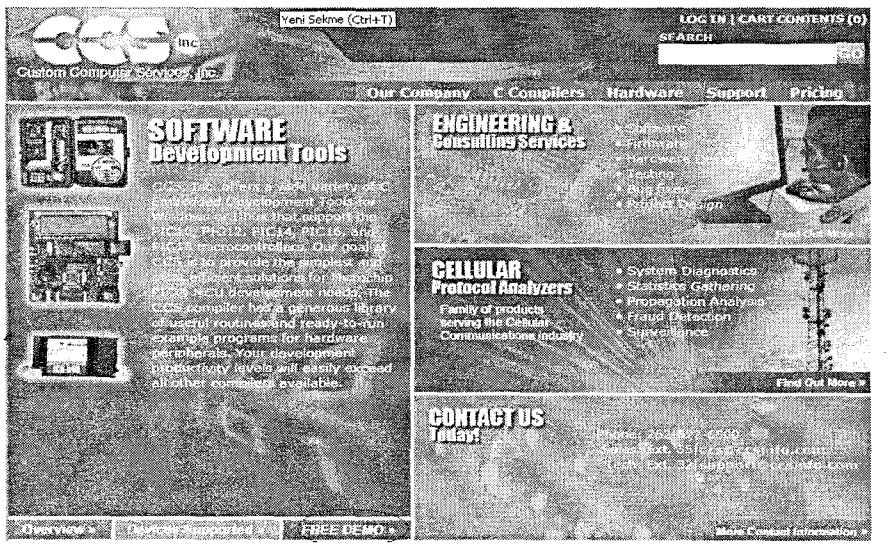
PIC16C52, PIC16F54, PIC16C54, PIC16C54A, PIC16CR54A, PIC16CR54B, PIC16C54B, PIC16C54C, PIC16CR54C, PIC16C55, PIC16C55A, PIC16C56, PIC16C56A, PIC16CR56A, PIC16C57, PIC16F57, PIC16C57A, PIC16CR57A, PIC16CR57B, PIC16C57B, PIC16CR57C, PIC16C57C, PIC16C58, PIC16CR58A, PIC16C58A, PIC16CR58B, PIC16C58B, PIC16F59, PIC16C61, PIC16C62, PIC16CR62, PIC16C62A, PIC16C62B, PIC16C63, PIC16CR63, PIC16C63A, PIC16CR64, PIC16C64, PIC16C64A, PIC16CR65, PIC16C65, PIC16C65A, PIC16C65B, PIC16C66, PIC16C67, PIC16C70, PIC16C71, PIC16C72, PIC16F72, PIC16CR72, PIC16C72A, PIC16C73, PIC16CR73, PIC16F73, PIC16C73A, PIC16C73B, PIC16C74, PIC16F74, PIC16CR74, PIC16C74A, PIC16C74B, PIC16LC74B, PIC16C76, PIC16F76, PIC16CR76, PIC16F77, PIC16C77, PIC16CR77, PIC16CR83, PIC16F83, PIC16C83, PIC16CR84, PIC16F84, PIC16C84, PIC16LF84A, PIC16F84A, PIC16C84A, PIC16F87, PIC16F88, PIC16C432, PIC16C433, PIC16C505, PIC16F505, PIC16F506, PIC16HV540, PIC16C554, PIC16C558, PIC16F610, PIC16HV610, PIC16HV616, PIC16F616, PIC16C620 ,PIC16C620A, PIC16CR620A, PIC16C621, PIC16C621A, PIC16C622, PIC16C622A, PIC16CE623, PIC16CE624, PIC16CE625, PIC16F627, PIC16F627A ,PIC16F628, PIC16F628A, PIC16F630, PIC16F631, PIC16F636, PIC16F639, PIC16C641, PIC16C642, PIC16F648A, PIC16C661, PIC16C662, PIC16F676, PIC16F677, PIC16F684, PIC16F685, PIC16F687, PIC16F688, PIC16F689, PIC16F690, PIC16C710, PIC16C711, PIC16C712, PIC16C715, PIC16F716, PIC16C716, PIC16C717, PIC16F737, PIC16C745, PIC16F747, PIC16C765, PIC16F767, PIC16C770, PIC16C771, PIC16C773, PIC16C774, PIC16F777, PIC16C781, PIC16C782, PIC16F785, PIC16HV785, PIC16F818, PIC16F819, PIC16F870, PIC16F871, PIC16F872, PIC16F873, PIC16F873A, PIC16F874, PIC16F874A, PIC16F876, PIC16F876A, PIC16F877, PIC16F877A, PIC16F883, PIC16F884, PIC16F886, PIC16F887, PIC16F913, PIC16F914, PIC16F916, PIC16F917, PIC16C923, PIC16C924, PIC16C925, PIC16C926, PIC16F946

PIC18F242, PIC18C242, PIC18F248, PIC18F252, PIC18C252, PIC18F258, PIC18F442, PIC18C442, PIC18F448, PIC18F452, PIC18C452, PIC18F458, PIC18C601, PIC18C658, PIC18C801, PIC18C858, PIC18F1220, PIC18F1230, PIC18F1320, PIC18F1330, PIC18F2220, PIC18F2221, PIC18F2320, PIC18F2321, PIC18F2331, PIC18F24J10, PIC18F2410, PIC18LF24J10, PIC18F2420, PIC18F2423, PIC18F2431, PIC18F2439, PIC18F2450, PIC18F2455, PIC18F2480, PIC18LF25J10, PIC18F25J10, PIC18F2510, PIC18F2515, PIC18F2520, PIC18F2523, PIC18F2525, PIC18F2539, PIC18F2550, PIC18F2580, PIC18F2585, PIC18F2610, PIC18F2620,

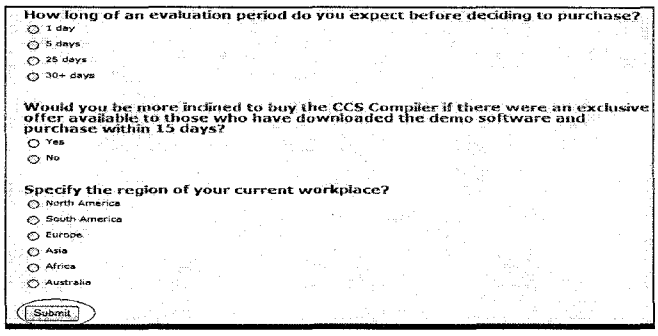
PIC18F2680, PIC18F2682, PIC18F2685, PIC18F4220, PIC18F4221, PIC18F4320, PIC18F4321, PIC18F4331, PIC18LF44J10, PIC18F44J10, PIC18F4410, PIC18F4420, PIC18F4423, PIC18F4431, PIC18F4439, PIC18F4450, PIC18F4455, PIC18F4480, PIC18F45J10, PIC18F4510, PIC18LF45J10, PIC18F4515, PIC18F4520, PIC18F4523, PIC18F4525, PIC18F4539, PIC18F4550, PIC18F4580, PIC18F4585, PIC18F4610, PIC18F4620, PIC18F4680, PIC18F4682, PIC18F4685, PIC18F6310, PIC18F6390, PIC18F6410, PIC18F6490, PIC18F65J10, PIC18F65J15, PIC18F6520, PIC18F6525, PIC18F6527, PIC18F6585, PIC18F66J10, PIC18F66J15, PIC18F6620, PIC18F6621, PIC18F6622, PIC18F6627, PIC18F6680, PIC18F67J10, PIC18F6720, PIC18F6722, PIC18F8310, PIC18F8390, PIC18F8410, PIC18F8490, PIC18F85J10, PIC18F85J15, PIC18F8520, PIC18F8525, PIC18F8527, PIC18F8585, PIC18F86J10, PIC18F86J15, PIC18F8620, PIC18F8621, PIC18F8622, PIC18F8627, PIC18F8680, PIC18F87J10, PIC18F8720, PIC18F8722

3.2. CCS C DEMO PROGRAMININ KURULMASI

http://www.ccsinfo.com adresinden CCS C derleyicisi satın alınabilir veya demo sürümü indirilebilir. Kitapta incelenen örneklerin çoğu CCS C derleyicisinin demo sürümünde derlenebilir. Fakat kitapta incelenen bazı uygulamaları CCS C DEMO sürümü desteklememektedir. Şekil-3.1'de CCS'nin ana sayfası görülmektedir. Bu sayfada **FREE DEMO** bağlantısına tıklanırsa önümüze doldurmamız gereken Şekil-3.2'deki gibi bir form gelecektir.

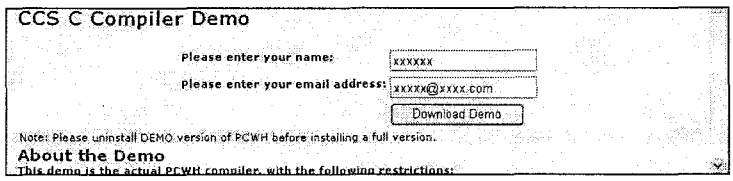


Şekil-3.1. CCS C Ana sayfası.



Şekil-3.2. *Demo sürümü için doldurulması gereken form.*

Bu formu doldurup **Submit** butonuna tıklandığında önümüze Şekil-3.3'te görüldüğü gibi ismimizin ve e-mail adresimizin yazılacağı kısım gelir. Bu bölümde doldurulduğunda **Download Demo** butonuna tıklandığında CCS C'nin 4 versiyonunun demo sürümü indirilebilir.



Şekil-3.3. *Demo sürümü için doldurulması gereken isim ve e-mail kısmı.*

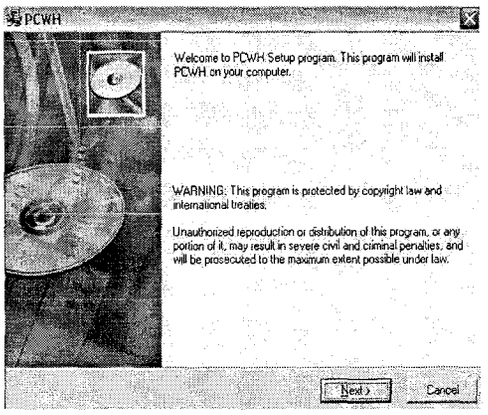
Demo sürümü sınırlı sayıda PIC ürününü desteklemekte ve CCS C derleyicisinin **çevresel birimler için** (LCD modülleri, keypad'ler, 24Cxx ve 94xx EEPROM'ları, DS1302 ve NJU6355 gerçek zamanlı saat entegreleri, DS2223 ve PCF8570 seri SRAM, sıcaklık sensörleri, dijital potansiyometreler vb.) **sağladığı onlarca hazır fonksiyon dosyalarını bulundurmamaktadır**. Ayrıca **demo sürümü en fazla 2Kb'lık programları derlemekte** ve **30 günlük bir kullanım süresi** sunmaktadır. Ayrıca kullanmak için internete bağlı olmak gerekmektedir. CCS C'nin **demo sürümünün desteklediği denetleyiciler** şunlardır;

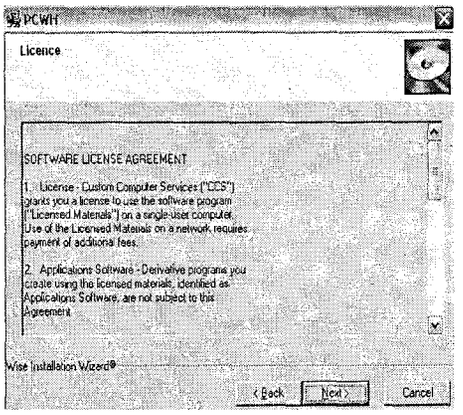
PIC16C554, PIC16F877, PIC16F877A, 18F458

Ayrıca CCS C ile ilgili diğer bilgilerde bu sitede bulunabilir. Ayrıca bu sitede mevcut olan forum sayfasında çok yararlı bilgiler bulunmaktadır. **http://www.ccsinfo.com/forum** adresinden CCS C forum sitesine girilebilir. Dünyadaki diğer tüm kullanıcılarla problemlerin paylaşıldığı ve kolaylıkla çözüm bulunduğu bir forum sayfasıdır. Forum sayfasına ücretsiz olarak üye olunabilir.

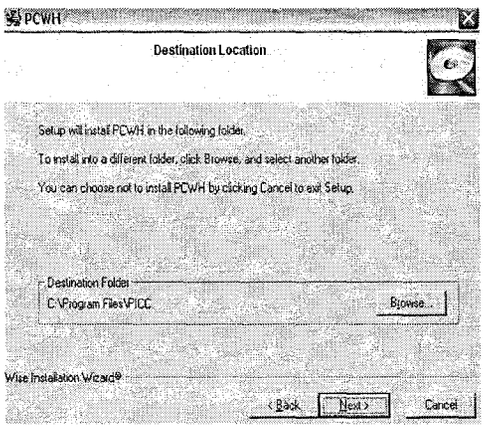
CCS C derleyicisinin demo versiyonu indirilip çalıştırıldığında karşımıza Şekil-3.4'teki gibi bir ekran gelecektir. Bu ekranda "Next" butonuna tıklandığında önümüze

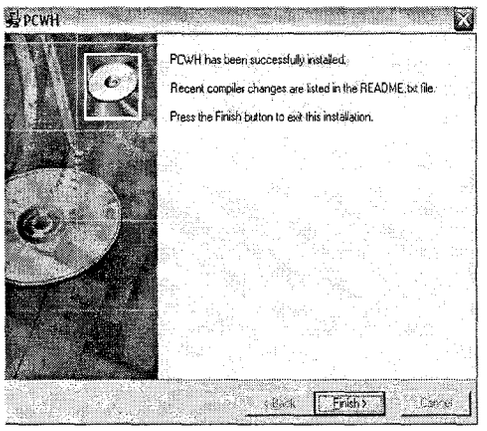
Şekil-3.5'ki gibi lisans sözleşmesinin olduğu ekran gelecektir. Orada da "Next" tuşu tıklandığında programın kurulacağı dizini gösteren ve/veya değiştirmemizi sağlayan menü ekrana gelecektir (Şekil-3.6). Next tuşuna bastıktan sonra programın windows sistemindeki grup ismi seçim penceresi ekrana gelir. Bu pencerede de Next tuşuna basıldıktan sonra program kurulmaya başlanır ve en son olarak da kurulumun bittiğini belirten pencere ekrana gelir (Şekil-3.7).





Şekil-3.4. *CCS C kurulum açılış ekranı.* ***Şekil-3.5.*** *CCS C lisans sözleşmesi ekranı*

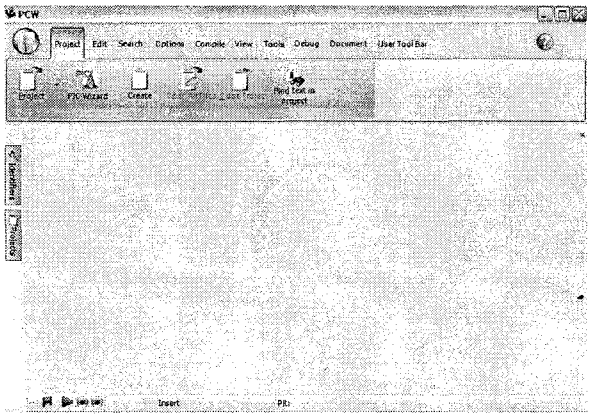




Şekil-3.6. *Program dizin seçim penceresi.* ***Şekil-3.7.*** *Kurulum bitiş ekranı.*

3.3. CCS C DERLEYİCİSİ MENÜLERİ

CCS C derleyicilerinin sürüm 3 serisi ile sürüm 4 serisinin ara yüzleri birbirinden farklıdır. Ayrıca sürüm 4 serisi CCS C derleyicilerinde sadece ara yüz değişikliği değil, derleme işlemlerinde olan bir çok hata giderilmiş ve derleme işlemi iyileştirilmiştir. Bu bölümde en son sürüm olan sürüm 4.xxx versiyonu anlatılmıştır.

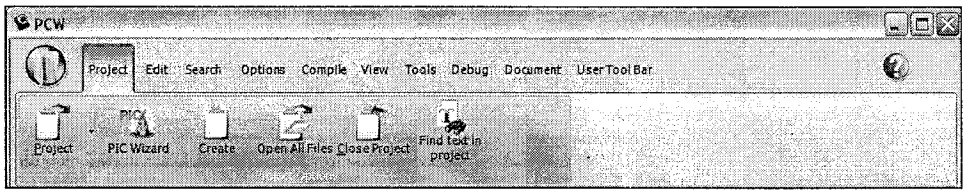


Şekil-3.8. *CCS C programının sürüm 4 serisinin ara yüz görüntüsü.*

3.3.1. File Menüsü

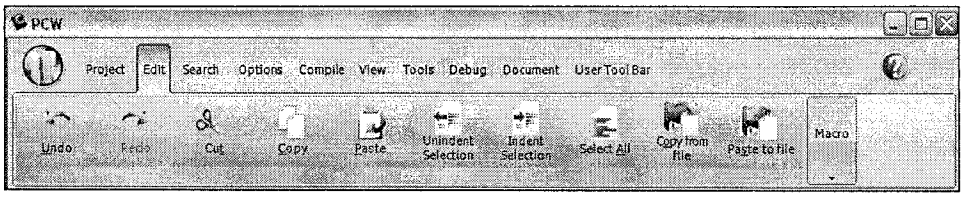
New	Yeni bir dosya açar.
Open	Önceden kaydedilmiş bir dosya açılır.
Close	Üzerinde çalışılan dosyayı kapatır.
Close All	Tüm açık dosyaları kapatır.
Save	Çalışmayı kaydeder.
Save As	Çalışmayı farklı bir isimde ve yere kaydeder.
Save All	Tüm açık dosyaları diske kaydeder.
Encrypt	Tanımlama dosyalarını (include files) şifreler.
Print	Çalışma yapılan dosyayı yazdırır.
Recent Files	Kullanıcı tarafından kullanılan en son dosyaları listeler.
Exit	Programdan çıkar.

3.3.2. Project Menüsü Şeridi



Project	Önceden kaydedilmiş bir proje dosyasını (.PJT) açar.
PIC Wizard	Otomatik olarak proje dosyası oluşturmamızı sağlar. Seçilen denetleyicinin özelliklerine göre istenen durumların seçilmesini ve bu seçimlerin kod karşılığını direkt olarak proje dosyasına ekler.
Create	Manuel olarak proje dosyası oluşturmamızı sağlar.
Open All Files	Proje dosyasına ait tüm dosyaları açar.
Close Project	Geçerli proje dosyasını tüm dosyaları ile birlikte kapatır.
Find Text In Project	Proje dosyasına ait tüm dosyaların içinde girilen metin veya karakteri arar.

3.3.3. Edit Menüsü Şeridi



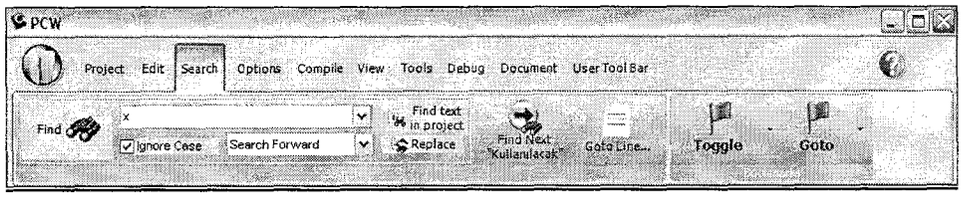
Undo	En son işlemi geri alır.
Redo	En son geri alınan işlemi eski haline getirir.
Cut	Kesme komutu.
Copy	Kopyalama komutu.
Paste	Yapıştırma komutu.
Unindent Selection	Bu komut seçildiğinde kursörün bulunduğu satırdan bir satır başı boşluğu geri alınır.
Indent Selection	Bu komut seçildiğinde kursörün bulunduğu satıra bir satır başı boşluğu verilir.
Select All	Tüm metni seçer.

Copy From File Seçilen dosyanın içindeki tüm metni kursörün bulunduğu yere kopyalar.

Paste to File Seçilen metni istenen dosyaya yapıştırır.

Macros Makro işlemlerinin (kayıt, tuş atama vb.) yapıldığı kısım.

3.3.4. Search Menüsü Şeridi



Find Yazılan metni veya karakteri dosyada arar.

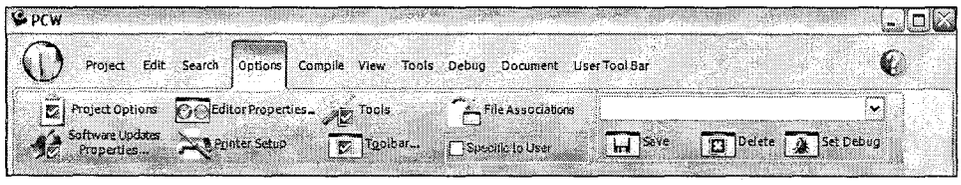
Find Text in Project Projede kayıtlı tüm dosyalar da istenen metni veya karakteri arar.

Goto Line İstenen numaralı satıra gider.

Toggle Bookmark 0-9 arası yer imi konulmasını sağlar.

Goto Bookmark Kursör seçilen yer imine gider.

3.3.5. Options Menüsü Şeridi



Project Options Proje dosyası ile ilgili ayarlamalar yapılır.

Editor Properties Derleyici programının genel ayarlarının, Görüntü ve yazı biçimlerinin ve renklendirilmesinin ayarları yapılır.

Tools Kullanıcının isteğine göre istenen menünün gösterilmesini ve ayarlarının yapılmasını, başka menü eklemelerinin eklenip çıkartılması gibi işlemler yapılır.

Software Updates Hangi programın güncelleneceğinin (update) belirlendiği,

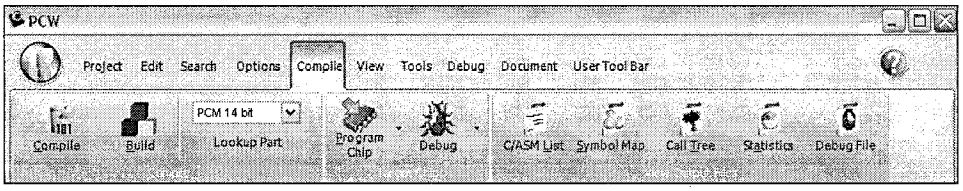
Properties hangi sıklıkla kullanıcıya güncelleme hatırlatılması yapılacağı gibi ayarlar yapılır.

Printer Setup Yazıcı ayarlarının yapıldığı kısım.

Toolbar Setup İstenen araç çubuğunun ve/veya simgelerinin (ikons) eklenip çıkartılması ve kullanıcıya özgü araç çubuğunun yaratılmasını sağlar.

File Associations Hangi dosyanın hangi program ile ilişkilendiğini, genel proje dosyalarının kayıt edildiği ve açıldığı dizinlerinin ayarlandığı ve program dizinlerinin belirlendiği yer.

3.3.6. Compile Menüsü Şeridi



Compile Üzerinde çalışılan projenin derlenmesini sağlayan komuttur. Bu işlem sonunda C dili ile yazılan program makine diline çevrilir. Bu işlemle PIC'e yüklenecek olan .HEX uzantılı dosya elde edilir.

Build Proje içindeki bir veya birden dosyayı derler.

Lookup Part Kullanılacak denetleyici modeli seçilir. Programda otomatik olarak seçilen denetleyiciye uygun derleyici programı seçer (PCM 12 bit, PCM 14 bit, PCM 16 bit).

Program Chip Denetleyicinin programlama kartı ile programlanması komutu.

Debug .hex giriş ve .asm çıkış dosyalarının hata ayıklanmasının yapılmasını sağlar.

C/ASM List Açık olan C diliyle yazılmış programın PIC Assembly komut karşılıklarını verir.

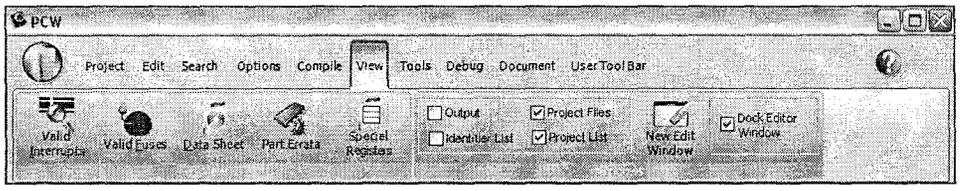
Symbol Map Sembol haritası programda kullanılan değişkenlerin kaydedicide nereye kaydedildiğini gösterir. Ayrıca derleyici ayarları ile derleme sonucunda yaratılan dosyaları ve bunların diskteki yolunu gösterir.

Call Tree Call tree komutu, programda kullanılan fonksiyonları ve fonksiyonların içindeki komutlarla bunların RAM veya ROM daki bilgilerini verir.

Statistics Derlenen program ile ilgili statiksel bilgileri verir.

Debug File Bu komut seçildiğinde derleme aşamasında oluşturulan kod tablosu dosyasını metin editöründe gösterir. Böylece .cof uzantılı dosyanın içeriğini görmüş oluruz.

3.3.7. View Menüsü Şeridi



Valid Interrupts Bu komut seçildiğinde gelen menüden kullanılan PIC ürünü seçildiğinde, alt tarafta seçilen PIC'in sahip olduğu kesmeler (interrupts) gelir. Böylece kullanıcı kullandığı denetleyicinin hangi kesmelere sahip olduğunu görebilir.

Valid Fuses Bu komut seçildiğinde gelen menüden kullanılan PIC ürünü seçilir ve seçim sonucunda alt tarafta seçilen PIC'e özgü programın başlangıç kısmında belirtilmesi gereken konfigürasyon ayarları gelir. Kullanılan osilatör çeşidi, Reset işlemleri, koruma gibi.

Data Sheet PIC ürünleri ile ilgili datasheet dosyalarını açar. Eğer yükleme esnasında bu dosyalar yüklenmediyse, kurulum CD sini ister.

Part Errata Kullanıcıya hata veri tabanını ve bilgilerini gösterir.

Special Registers Seçilen denetleyici ile ilgili özel fonksiyon kaydedicilerini gösterir.

New Edit Windows Yeni pencerede o an çalışılan dosyanın yan yana istenen yere döşenmesini sağlar.

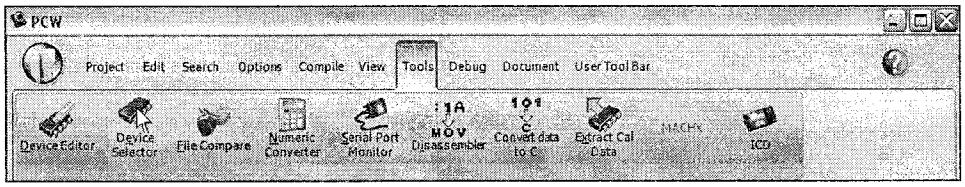
Project Files Bu seçme kutusu seçildiğinde Project dosyaları açılmış olarak ekranda görünür.

Project List Bu seçme kutusu seçildiğinde açılan proje listeleri açılmış olarak ekranda görünür.

Output Bu seçme kutusu seçildiğinde, yazılan programın derleme sonucunda oluşan uyarı (warning) ve hata (error) bilgileri menüsü ekranın alt tarafında görünür.

Identifier List Bu seçme kutusu seçildiğinde, program tarafından tanımlanan tanımlar penceresi görüntülenir.

3.3.8. Tools Menüsü Şeridi



Device Editor Bu komut ile PIC ürünlerinin derleme aşamasında kullanılacak bazı ayarlarının değişimi yapılabilir. Listeden istenen PIC ürünü seçilerek "Edit" komutuna basıldığında ayarlar menüsü ekrana gelir ve istenen değişiklik yapılabilir. Böylece istenen bir ürün güncellemesi yapılmış olur.

Device Selector Bu komut ile gelen menüde istenen özellikler seçilir ve bu özelliklere uyan ürünleri kolayca bulunabilir.

File Compare Bu komut ile seçilen iki kaynak veya metin dosyası karşılaştırılır.

Numeric Converter Bu komut seçildiğinde desimal, hex, float, unsigned, signed gibi tipteki değerlerin birbirleri ile karşılıklarını verir.

Serial Port Monitör Bu komut ile yapılan çalışmada RS232 iletişimi kullanılmışsa programın denemesi yapılabilir. Yani seri iletişim benzetim programını açar.

Disassembler Bu komut seçilen .HEX uzantılı dosyadan bu kodların karşılığı olan .ASM dosyasını elde eder. Böylece kullanıcının elinde sadece .hex kodları bulunuyorsa bu komut kullanıcıya bu kodların assembler karşılığını vererek program üzerinde değişiklik yapma fırsatı verir.

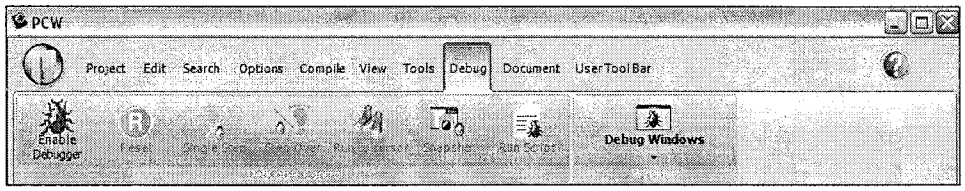
Convert Data to C Seçilen giriş dosyasını (binary, hex, integer, float formatındaki) istenen formda (#ROM, CONST) kod dönüşümünü verir.

Extract Calibration Bu komut .hex dosyasının içindeki C başlık dosyasına gidecek ayar bilgilerini ayıklar.

Mach-X Mach-X.exe programını çağırır ve o an projede kullanılan denetleyicinin hex dosyasını internetten indirir.

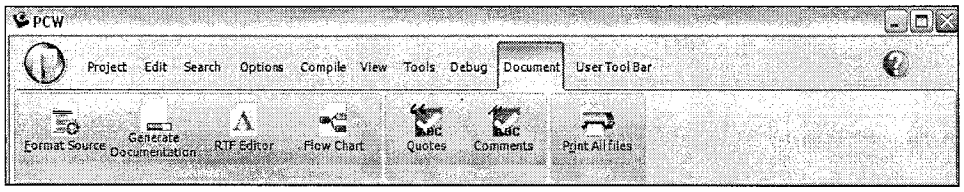
ICD ICD.exe programını çağırır ve o an projede kullanılan denetleyicinin hex dosyasını internetten indirir.

3.3.9. Debug Menüsü Şeridi



Enable Debugger	Hata ayıklayıcısını çalıştırır ve hata ayıklama penceresini açar.
Reset	Hata ayıklayıcısını resetler.
Single Step	Tek bir kaynak kodunu işleme sokar.
Step Over	Tasarım alanında bulunan devreyi adım adım çalıştırır.
Run to Cursor	Kursörün bulunduğu kodu çalıştırır.
Snapshot	Kullanıcıların değişik hata ayıklama bilgilerini kaydetmesini sağlar.
Debug Windows	Bu açılır pencere belirli hata ayıklama bilgilerinin görülmesini sağlar.

3.3.10. Document Menüsü Şeridi

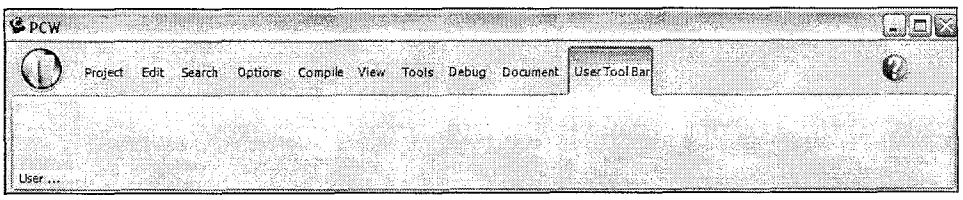


Format Source	Kaynak dosyanın girinti, bazı karakter ve kelimelerin renklendirilmesi gibi biçimlendirme ayarları yapılır.
Generate Documentation	Kaynak kod derleme bilgilerini .rtf dosyası biçiminde verir.
RTF Editor	RTF editör programını açar.
Flow Chart	Akış diyagram editör programını açar. Bu program ile yazılan programın akış şeması çizilebilir. Aynı zamanda bu program ile elektriksel devrede çizilerek programa bilgi maiyetinde eklenebilir. Program sonunda kaydedilen dosyanın uzantısı .dgr olur.
Qutoes	Tırnak içindeki tüm kelimelerin sözcük kontrolünü yapar.

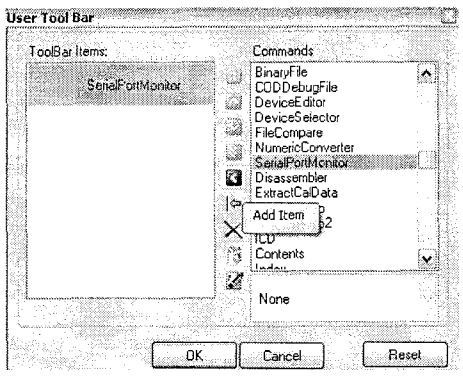
Comments Kaynak dosya içindeki tüm kelimelerin sözcük kontrolünü yapar.

Print All Files Üzerinde çalışılan projeye bağlı tüm dosyaları yazdırır.

3.3.11. User ToolBar Menüsü Şeridi

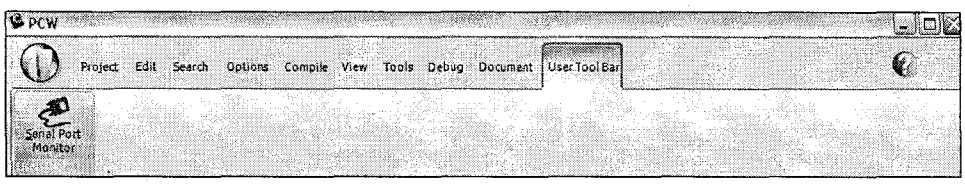


Bu menü kullanıcıya özel bir menü şeridididir. Bu menüye kullanıcı istediği komut butonlarını ekleyebilir. Bu menü seçildiğinde "User..." yazan yazının sağ alt kısmında ki kutu tıklandığında aşağıda Şekil-3.9'da verilen menü önümüze gelir.



Şekil-3.9. *User Tool Bar menüsü.*

Bu menüde sağ tarafta ki kısımdan istenen komut seçilir ve orta şeritteki kırmızı "Add Item" butonuna tıklandığında seçilen bu komut sol tarafa geçer. Böylece istediğimiz bir komutu kendi menümüze eklemiş oluruz. Ekleme işlemi sonucunda menü Şekil-3.10'da verildiği gibi olur. Eğer eklenen bir buton silinmek isteniyorsa, eklenmiş buton komutu seçilir ve orta şeritte kırmızı çarpı şeklinde ki butona "Delete Item" tıklanarak silme işlemi gerçekleştirilir.



Şekil-3.10. *User Tool Bar menüsüne buton eklenmiş durumu.*

3.3.12. Help Menüsü

Contents	Yardım dosyasının içeriğini gösterir.
Index	Yardım dosyasının içindeki tüm dizinleri gösterir ve arama imkanı verir.
Keyword at Cursor	Bu komut seçilmeden önce kursör ile yardım alınacak bir komut seçilir. (örneğin #define komutu seçilir) ve bu komut seçilirse (ya da F1'e basılırsa) bu komut ile ilgili yardım gelir.
Debugger Help	Hata ayıklama ile ilgili yardım.
Editor	Derleyicide kullanılan kısa yol tuşlarını gösterir.
Data types	Bu komut seçildiğinde derleyicide kullanılabilen data tipleri ekrana gelir.
Operators	Bu komut seçildiğinde derleyicide kullanılabilecek C operatörleri ekrana gelir.
Statements	Bu komut seçildiğimde ekrana derleyicide kullanılabilecek döngü fonksiyonları gelir.
Preprocessor Cmds	Bu komut seçildiğinde CCS C derleyicisinde kullanılan ön işlemci komutları açılır.
Buit in Functions	Bu komut seçildiğinde CCS C derleyicisinde var olan fonksiyonların listesi ekrana gelir.
Technical Support	Bu komut ile teknik destek almak için üreticiyi firmaya elektronik posta gönderilebilir.
Check for Software Updates	Kullanılan versiyonun güncel sürümü olup olmadığını kontrol eder.
Internet	CCS C programının ana sitesine ve diğer bazı ilgili sitelere direkt erişim sağlar.
About	Programın üreticisi ve sürümü hakkında bilgi verir.

4. Bölüm

CCS C Derleyicisi Temel Kavramları ve Fonksiyonları

- Tanıtıcı Kelimeler
- C'de Program Satırları
- Değişken Tipleri ve Tanımlamaları
- Yer Belirleyicileri
- Değişken Tip isimlerine Takma İsim (Alias) Verme
- Sabit Tanımlama ve ROM Belleğe Bilgi Yazma
- CCS C Derleyicisinde Kullanılan Operatörler
- CCS C'de Özel Karakterlerin Gösterimi
- Dizi Tanımlama
- Göstericiler (Pointers)
- Yapılar (Structures)
- CCS C'de Fonksiyon Tanımlama
- Açıklayıcı Yazımı
- CCS C Programı Genel Gövdesi
- CCS C Komut Seti
- CCS C Ön İşlemci Fonksiyonları
- CCS C Mikrodenetleyici Kontrol Fonksiyonları
- CCS C Karakter Fonksiyonları
- CCS C Matematik Fonksiyonları

C programlama dili, 1970'li yılların başında **Dennis Ritchie** tarafından **AT&T** laboratuarlarında geliştirilen bir programlama dilidir. Ritchie o zamanlar B adlı programlama dilinin tasarımcısı **Ken Thomson** ile birlikte UNIX işletim sistemi üzerinde çalışıyordu. Ritchie B diline yeni fikirler getirerek *C* dilinin temellerini atmış oldu. Bu nedenle denilebilir ki C dilinin temeli B diline dayanmaktadır. C dili gelişim evresinin doruk noktaları 1972 yılında yoğunlaştığı için C dilinin doğumu 1972 yılı olarak kabul edilebilir. C dili doğumunun ilk zamanlarında UNIX sistemler için geliştirilmiş olmasına ve UNIX tabanlı sistemler ile anılmasına rağmen günümüzde kişisel bilgisayarların yaygınlaşmasıyla en fazla tercih edilen programlama dilleri arasında göze çarpmıştır. Bugün neredeyse her tür sistemde C derleyicilerine rastlamak mümkündür. C, 1983 yılında ANSI (**A**merican **N**ational **S**tandarts **I**nstitute-Amerikan Ulusal Standart Enstitüsü) tarafından standardize edildikten sonra yüksek oranda taşınabilir bir sistem programlama dili olmuştur. Günümüzde

işletim sistemleri, derleyiciler, editörler gibi sistem programlarının çoğunun yazım dilini C veya C temelli programlar (C++, C# vb.) oluşturmaktadır.

CCS C derleyicisi ANSI C'nin neredeyse tüm temel tanımlamalarını kabul eder. Diğer C tabanlı PIC derleyici programlarına nazaran CCS C, C diliyle neredeyse komple uyum içindedir. Bu bölümde CCS C'nin temel kavramları ve yapısı üzerine bilgiler verilecektir.

4.1. TANITICI KELİMELER

Program içerisinde kullanılan değişken, sabit, fonksiyon gibi isimleri için kullanılan ve programcı tarafından belirlenen özel isimlerdir. Tanıtıcı kelimeler aşağıda verilen kurallara uygun olmalıdır.

1- Harf veya _ (altçizgi karakteri) ile başlamalıdır.
(sayi, ilksay, x, AD, şehir...) (doğru)
2- İlk karakterden sonra rakam veya altçizgi karakterleri kullanılabilir.
(ilk_sayi, ad_soyad, xl, say35...) (doğru)
3- İçerisinde Türkçe karakter, boşluk ve özel karakter bulunamaz.
(sayı, bilgi yaz, damga-vergisi, ...) (yanlış)
4- Bir C kelimesi (keyword) tanıtıcı kelime olamaz.
(long, int, break, for, while,... vb.) (yanlış)
5- Tanıtıcı kelimelerde küçük harf ve büyük harf ayrımı yapılmaktadır. **X** ile **x** değişkenleri aynı değildir. Aynı şekilde **bilgi** ve **Bilgi** değişkenleri de aynı değildir. Bu ifadelerin her biri farklı hafıza alanlarına karşılık gelir.

4.2. C'DE PROGRAM SATIRLARI

C dilinde her komut noktalı virgül (;) karakteri ile sonlandırılır. C programlama dilinde satır kavramı yoktur. Komutlar, ifadeler, fonksiyonlar, tanımlamalar vb. yan yana yazılabileceği gibi alt alta da yazılabilir.

```
for (x=0;x<10;x++)  printf ("%d \n",x);
```
ifadesi

```
for
{
x=0;
      x<
10;x++)
printf ("%d \n",x
}     ;
```
gibi de yazılabilir.

Ancak programın okunabilirliğini zorlaştıran yazım şekillerinden kaçınılmalıdır.

4.3. DEĞİŞKEN TİPLERİ VE TANIMLAMALARI

C programlama dilinde kullanılan temel değişken tipleri **char, int, float** ve **double**'dır. Ancak bu veri tiplerinin önüne **short (kısa), unsigned (işaretsiz), signed (işaretli)** ve **long (uzun)** kelimeleri kullanılarak değişik veri tipleri meydana getirilir. Bunlara **tip değiştiricileri** denir. CCS C de ise hem değişken tipleri ile hem de sadece tip değiştiricileri ile değişken tipi tanımlaması yapılabilir. Tablo-4.1'de CCS C'deki değişken tipleri görülmektedir.

Değişken Tipi	Açıklama	En Küçük	En Büyük
int1	1 bit'lik tam sayıyı ifade eder	0	1
int8	8 bit'lik tam sayıyı ifade eder	-128	+128
int	int8 ile aynı anlamdadır	-128	+128
unsigned int	İşaretsiz tam sayı	0	255
int16	16 bit'lik tam sayıyı ifade eder	-32768	+32768
unsigned int16	İşaretsiz 16 bit'lik tam sayı	0	65536
int32	32 bit'lik tam sayıyı ifade eder	-2147483648	+2147483648
char	8 bit'lik karakteri ifade eder	0	255
float	32 bit'lik ondalıklı sayıyı ifade eder		
short	int1 ile aynı anlamdadır		
long	int16 ile aynı anlamdadır		
void	sayının belirli bir tanımının olmadığını belirtir		

Tablo-4.1. *CCS C'de değişken tipleri.*

```
int a,b ;       // a ve b adında 8 bit'lik değişkenler tanımlanıyor.
int8 p;         // p adında 8 bit'lik bi değişken tanımlanıyor.
float x;        // x adında bir ondalıklı değişken tanımlanıyor.
char  i;        // Karakter türünde i adında bir değişken tanımlanıyor.
int16 bilgi;    // bilgi adında 16 bit'lik tamsayı tipinde bir değişken tanımlanıyor.
```

CCS C derleyicisinde sayıların desimal, heksadesimal, binary vb. olduğunu belirtmek gerekir. Aşağıda Tablo-4.2'de bu gösterimlerin nasıl olacağı verilmiştir.

Data Gösterimi	Açıklama
12	Desimal
012	Oktal
0x12	Heksadesimal
0b00010010	Binary
'x'	Karakter
'**0**10'	Oktal karakter
'**x**A5'	Hex karakter
"abcdef"	String ifade

Tablo-4.2. *Data tanımlamaları.*

Sayının önüne hiçbir şey yazılmazsa **desimal** olarak algılanır. **0x** eki sayının önüne getirilirse **heksadesimal** sayı olarak, **0b** eki sayının önüne getirilirse **binary** olarak algılanır. **Karakter** tanımlamaları için, karakterler tek tırnak ' ' içine yazılarak belirtilir. **String** ifade tanımlanırken ise çift tırnak " " kullanılır.

```
x=253;          // x değişkeni içeriği desimal olarak 253 olur.
i=0xF3;         // i değişkeni içeriği heksadesimal olarak F3'tür.
data="GAZI";    // data değişkeni içeriği string olarak "GAZI" kelimesidir.
port=0b10010001; // port değişkeni içeriği binary 10010001 değeridir.
```

4.4. YER BELİRLEYİCİLERİ VE TANIMLAMALARI

C dilinde yer belirleyicileri, nesnelerin nerede tutulmak istendiğini derleyici programa bildirmek için kullanılır. Örneğin programda tanımlanan bir değişkenin ne zaman oluşturulup ne zamana kadar geçerli olacağı programcı tarafından kontrol edilebilir. Bu durumda yer belirleyicilerin kullanılması gerekmektedir.

"auto" yer belirleyicisi = Yerel değişkenler için kullanılır. Bu sözcük değişken tanımlanmasının önüne getirildiğinde, değişkenin bulunduğu blok gelince değişkenin oluşturulacağı ve blok sonunda değişkenin bellekten silineceği anlamına gelir. Normalde değişkenler önüne hiçbir yer belirleyicisi konulmasa bile **auto** yer belirleyicisi takısını almış kabul edilir. auto ile tanımlanan değişkenler ile yazılan programlarda belleğin boşa kullanımı önlenmiş olunur.

```
{
        output_high(pin_C1);
        {
                auto int x ;        // Blok başı, x değişkeni yaratılır.
                ...
        }                           // Blok sonu, x değişkeni bellekten silinir.
}
```

"static" yer belirleyicisi = static yer belirleyicisi değişken tanımlamasının önüne getirildiği zaman, o değişkenin programın çalışması süresince bellekten silinmeyeceği anlamına gelir. Bir bakıma auto belirleyicisinin zıt anlamlısıdır.

"register" yer belirleyicisi = CCS C'de bu yer belirleyicisi kullanıldığında değişkene herhangi bir etkisi olmaz.

"extern" yer belirleyicisi = Birçok dosyadan (modülden) oluşan programlarda ana program dosyasında diğer dosyalar tanıtılır. Ana program dosyasına tanıtılan bu dosyaların içinde tanımlanmış değişkenler vardır ve bu değişkenler ana programda da tanıtılırsa bir karışıklık ortaya çıkacaktır. Bu nedenle bu yer belirleyicisi ilgili değişkenin kendi ana program dosyasında değil de, ana dosyaya eklenen diğer dosyalarda tanımlanmış olduğunu belirtmek için kullanılır.

4.5. DEĞİŞKEN TİP İSİMLERİNE TAKMA İSİM VERME (ALIAS)

CCS C'de değişken tip isimlerine programcı kendine göre kendi aklında daha kalıcı değişken tip ismi verebilir. Örneğin tam sayı değişkenlerinin tanımlamak için kullanılan *int* değişken tipi programcı tarafından *tamsayi* isminde kullanılmak istenebilir. Takma isim vermek için aşağıdaki komut kullanılmalıdır.

typedef [değişken tipi] [yeni değişken tipi ismi] ;

```
typedef int tamsayi;   // tamsayi kelimesi int değişken tipi görevini alıyor
tamsayi x;             // x isminde tam sayı tipinde bir değişken tanımlanıyor
```

Örnekte görüldüğü gibi *int* değişken tipinin görevini yapacak *tamsayi* adında bir değişken tipi tanımlanıyor. Daha sonra programda *int* yerine *tamsayi* ifadesi kullanılabiliniyor. İstenirse programda bu komuttan sonra int ifadesi de yine kullanılabilir.

4.6. SABİT TANIMLAMA VE ROM BELLEĞE BİLGİ YAZMA

Sabitler programın başında tanımlanırlar ve program içinde değiştirilemezler. Sabitler program belleği olan ROM hafızasına yerleştirilirler. Sabit tanımlamak için CCS C'de **const** anahtar sözcüğü kullanılır.

```
const int i=40;   // tam sayı tipinde i içeriği 40 olan i sabiti tanımlanıyor.
const char bilgi[5]={1,2,3,4,5};  // bilgi isminde karakter tipinde sabit
                                     dataya sahip bir dizi tanımlanıyor.
```

Aynı şekilde **#ROM** anahtar sözcüğü de ROM belleğe bir veri yazmak için kullanılır. Örnekte ifadenin sonuna noktalı virgül konulmadığına dikkat ediniz.

```
#rom 0x1002={9,8,7,6,5}
```

Bu komutla, ROM belleğe 0x1002 adresinden itibaren bilgiler yerleştiriliyor. Örneğin ROM'da 0x1002 adresinde 9, 0x1003 adresinde 8 bilgisi yer alır.

4.6.1. BackSlash (Ters Eğik Çizgi-\) Sabitleri

C programlama dilinde, backslash(\) kullanılarak bazı işlemlerin program içerisinde tanımlanabilmesi mümkündür. Tablo-4.3'de backslash (\) kullanılarak yapılabilen işlemler sıralanmıştır. İleriki bölümlerde incelenen fonksiyonlarda backslash

sabitlerinin nerelerde kullanıldığı daha iyi anlaşılacaktır. Özellikle seri iletişim komutlarında ve LCD ile ilgili komutlarda bu sabitler kullanılmaktadır.

Kod	Anlamı
\b	backspace
\n	bir sonraki satıra geç
\r	enter
\t	tab
\0	NULL
\a	zil sesi

Tablo-4.3. *BackSlash sabitleri.*

4.6.2. % Sabitleri

BackSlash (\) ile tanımlanan işlemlerin yanında özellikle ekrana bir şey yazdırırken yazılan karakterin veya kullanıcının yazdığı bir karakteri ekrandan alırken alınan karakterin hangi özellikte olacağını belirtmek için kullanılır. Özellikle seri iletişim komutlarında ve LCD ile ilgili komutlarda bu sabitler kullanılmaktadır.

Kod	Anlamı
%d	char ve int türlerini desimal sistemde
%c	Karakter görüntüsü biçiminde
%x	int türünü Hex sistemde (Harfler küçük yazılır)
%X	int türünü Hex sistemde (Harfler büyük yazılır)
%o	int türünü Octal sistemde
%u	unsigned int türünü desimal sistemde
%ld	long türünü desimal sistemde
%lx	long türünü Heksadesimal sistemde
%f	float ve double türlerini desimal sistemde
%e	float ve double türlerini üstel biçimde
%s	Stringleri karakter olarak

Tablo-4.4. *% Sabitleri.*

4.7. CCS C DERLEYİCİSİNDE KULLANILAN OPERATÖRLER

Operatörler, nesneler veya sabitler üzerinde önceden tanımlanmış bir takım işlemleri yapan atomlardır. Operatörler; Aritmetiksel operatörler, İlişkisel operatörler, Mantıksal operatörler, Bit operatörleri, Unary operatörler ve Özel

amaçlı operatörler olmak üzere altı grupta toplanabilir. Derleyicinin tanıdığı tüm operatörler ve açıklamaları Tablo-4.5'de ayrıntılı olarak verilmiştir.

Operatör	Açıklama
+	Toplama Operatörü.
+=	x+=y ifadesi x=x+y ile aynı anlamdadır.
&=	x&=y ifadesi x=x&y ile aynı anlamdadır.
&	Adres operatörü.
^=	x^=y ifadesi x=x^y ile aynı anlamdadır.
^	Özel VEYA
|=	x|=y ifadesi x=x|y ile aynı anlamdadır.
|	VEYA operatörü
--	Bir azaltma işlemi yapar
/=	x/=y ifadesi x=x/y ile aynı anlamdadır.
/	Bölme operatörü.
==	Eşitlik operatörü.
>	Büyüktür operatörü.
>=	Büyük veya eşittir operatörü.
++	Bir arttır işlemi yapar.
!=	Eşit değil operatörü.
<<=	x<<=y ifadesi x=x<<y ile aynı anlamdadır.
<	Küçüktür operatörü.
<<	Sola kaydırma operatörü.
<=	Küçük veya eşit operatörü.
&&	Mantıksal AND operatörü.
!	Mantıksal DEĞİL operatörü.
||	Mantıksal VEYA operatörü.
%=	x%=y ifadesi x=x%y ile aynı anlamdadır.
%	Bölme işlemi sonucunda kalanı verir.
*=	x*=y ifadesi x=x*y ile aynı anlamdadır.
*	Çarpma operatörüdür.
~	Tümleyenini alma operatörü.
>>=	x>>=y ifadesi x=x>>y ile aynı anlamdadır.
>>	Sağa kaydırma operatörü.
-=	x-=y ifadesi x=x-y ile aynı anlamdadır.
-	Çıkarma operatörü.

Tablo-4.5. *CCS C'de kullanılan operatörler.*

4.7.1. Aritmetiksel Operatörler

Aşağıda CCS C programlama dilinde kullanılan aritmetiksel operatörler verilmiştir. Toplama, çıkarma, çarpma ve bölme işlemleri kendi bilinen işlemlerini yaparlar. Modülüs bölme (%) ise iki sayının bölümünden kalanı verir. Örneğin, 5%2=1 olur.

Operatör	İşlem
-	Çıkarma
+	Toplama
*	Çarpma
/	Bölme
%	Modülüs bölme

Tablo-4.6. *Aritmetiksel operatörler.*

4.7.2. İlişkisel Operatörler

İlişkisel operatörler iki değer arasında büyüklük, küçüklük ve eşitlik kontrolü yaparlar.

Operatörler	Görevi
>	Büyüktür
>=	Büyük veya eşittir
<	Küçüktür
<=	Küçük veya eşittir
==	Eşittir
!=	Eşit değildir

Tablo-4.7. *İlişkisel operatörler.*

İlişkisel operatörlerle kurulan şart cümlesi doğru ise "1", yanlış ise "0" değeri üretilir.

```
a = 9;
b = 4;
if  (a>b)  printf ("a, b'den büyüktür");
```

Yukarıdaki örnekte a=9 ve b=4 olarak atanmıştır. Şart cümlesinde (a>b) doğru olacağı için 1 üretilir ve parantez içinin değeri 1 olur. Parantez içinin değeri doğru olduğu için onu takip eden komut işlem görür. Eğer şart doğru olmasaydı yani a>b olmasaydı parantez içinin değeri 0 olurdu ve onu takip eden komut işlem görmezdi. Aşağıda ilişkisel operatörlerle yapılan işlemler ve aldıkları sonuçlar görülmektedir.

```
a = (5 >2);        // 5, 2'den büyüktür. Şart doğrudur ve a = 1 olur.
c = (4 <= 1);      // 4, 1'den küçük veya eşit değildir. Şart yanlıştır
                      ve c=0 olur.
b = (3 == 3);      // 3 sayısı 3'e eşittir. Şart doğrudur ve b=1 olur.
```

4.7.3. Mantıksal Operatörler

Mantıksal operatörler birden fazla şart cümlesinin birbirlerine bağlanması işleminde kullanılır. CCS C derleyicisinde kullanılan mantıksal operatörler ve görevi aşağıda verilmiştir.

Operatörler	Görevi
&&	VE(AND)
||	VEYA(OR)
!	DEĞİL(NOT)

Tablo-4.8. *Mantıksal operatörler.*

CCS C'de ilişkisel bir operatörün ürettiği değer, koşulun sağlanması durumuna göre 1 (Doğru) ya da 0 (Yanlış) sayılarıdır. Fakat bir sayı mantıksal olarak yorumlanacağı zaman şu kural geçerlidir.
Sayı sıfır ise YANLIŞ(0) değeri üretilir. Sayı sıfır dışı bir değer ise, yani negatif ya da pozitif herhangi bir sayı ise DOĞRU(1) değeri üretilir.

Aşağıdaki sayıların mantıksal yorumlarını inceleyelim;

-14 Doğru
0 Yanlış
+98 Doğru

Mantıksal operatörler dijital sistemlerdeki görevlerini icra ederler. Aşağıda mantıksal operatörlerin hangi durumda hangi değerleri ürettikleri rahatça görülmektedir.

a	b	a **VE** b	a **VEYA** b
Yanlış(0)	Yanlış(0)	Yanlış(0)	Yanlış(0)
Yanlış(0)	Doğru(1)	Yanlış(0)	Doğru(1)
Doğru(1)	Yanlış(0)	Yanlış(0)	Doğru(1)
Doğru(1)	Doğr(1)	Doğru(1)	Doğru(1)

a	**DEĞİL** a
Doğru(1)	Yanlış(0)
Yanlış(0)	Doğru(1)

Aşağıda VE, VEYA ve DEĞİL mantıksal operatörlere örnekler verilmiştir.

```
a = 4 && 0;        // a = 0 olur
b = 10 && -4       // b = 1 olur
a = 3 || 0         // a = 1 olur
b = 0 || -30       // b = 1 olur
a = !6             // a = 0 olur
```

4.7.4. Bit Operatörleri

Bit operatörleri bit düzeyinde işlem yapan operatörlerdir; yani sayıları bir bütün olarak değil, bit bit ele alarak işleme sokarlar. Bit operatörlerin aldığı değişkenlerin

(operand'ların) türü tamsayı türlerinden olması gerekir (char, short, int, long), "float" ya da "double" türleri bu operatörlerle işleme sokulamaz.

Bit operatörü	Görevi
~	Bit değil (bitwise not)
<<	Sola öteleme (bitwise left)
>>	Sağa öteleme (bitwise right)
&	Bit VE (bitwise AND)
^	Bit ÖZEL VEYA (bitwise EXOR)
|	Bit VEYA (bitwise OR)

Tablo-4.9. *Bit operatörleri.*

4.7.4.1. Bit Değil (~) Operatörü

Tek operand alan ve her zaman önek durumunda olan bu operatör, 1'e tümleme işlemi yapar. Yani operand'ı olan sayıdaki 1 olan bit'leri 0; 0 olan bit'leri 1'e dönüştürür. Örneğin;

```
unsigned int x = 0x1834;  // işaretsiz tamsayı olarak x tanımlanıyor, 0x
                             heksadesimal anlamında
unsigned int y;           // işaretsiz tamsayı tipi olarak y tanımlanıyor
..............
y = ~x ;          // x'in bit düzeyinde değili alınıp y değişkenine aktarılıyor
```

Yukarıdaki örneği açıklayalım: Önce x içerisinde heksadesimal sistemde verilen sayıyı binary sisteme (bit düzeyine) dönüştürelim.

```
x = 0001 1000 0011 0100
    (1)  (8)  (3)  (4)
```

Şimdi x'in bit olarak değilini alalım. Bu işlem için 1'leri 0; 0'ları 1 yapmalıyız.

```
~x = 1110 0111 1100 1011
     (E)  (7)  (C)  (B)   sonucu elde edilir.
```

4.7.4.2. Bit VE (&) Operatörü

Karşılıklı bit'leri VE işlemine tabi tutar. VE işleminin doğruluk tablosunu mantıksal operatörleri anlatırken vermiştik. Bit VE işlemi mantıksal çarpma yapar. Aşağıdaki örneği inceleyelim.

```
x = 0xBB;
y = 0x1C;
........
z = x & y;   // x ve y değişkeni VE işlemine tabi tutulunca z değişkenin
                 değeri soruluyor
```

x ve y değişkenlerinin bit durumlarını inceleyelim.

x = 1011 1000 y = 0001 1100
(B) (8) (1) (C)

Karşılıklı bit'ler üzerinde VE işlemi yapılırsa aşağıdaki sonuç elde edilir.

```
 1011 1000   x
 0001 1100   y
 ------------
 0001 1000   x & y
```

4.7.4.3. Bit VEYA (|) Operatörü

Karşılıklı bit'ler üzerinde VEYA işlemini uygular. VEYA işleminin doğruluk tablosu ilişkisel operatörler kısmında verilmişti. Bit VEYA mantıksal toplama işlemi yapar.

```
        x = 0001 1111 1100 0000 (1FC0)
        y = 0001 0101 0011 1010 (153A)
z = x | y = 0001 1111 1111 1010  olur.
```

4.7.4.4. Bit ÖZEL VEYA (^) Operatörü

Bu operand karşılıklı operandları aynı ise 0, farklı ise 1 değerini üretir.

```
 0001 0101 1100 0100   a
 0001 0010 1100 0101   b
 ----------------------------
 0000 0111 0000 0001   a ^ b
```

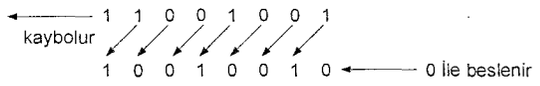
4.7.4.5. Sola Öteleme (<<) Operatörü

Sola öteleme operatörünün sol tarafındaki operand öteleme işleminin yapılacağı değişkeni ya da sabiti, sağ tarafındaki operand ise öteleme sayısını göstermektedir.

değişken << öteleme sayısı

Sola öteleme işleminde bütün bit'ler öteleme sayısı kadar sola kaydırılır. Kaydırma işlemi sonunda en sağdaki bit sürekli sıfırla beslenir, en soldaki bit ise menzil dışına çıktığı için kaybolur. Örneğin;

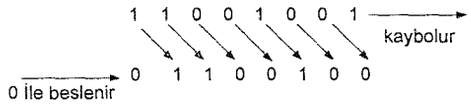
```
a = 0xC9      // binary olarak 1100 1001 değerine eşittir.
b = a << 1;   // a'nın 1 kere sola öteleneceğini belirtir. b=0x92 olur.
```



4.7.4.6. Sağa Öteleme (>>) Operatörü

Sağa ötelemenin sola ötelemeden farkı yönünün ters olmasıdır. Sağa ötelemede sayı içerisindeki bütün bit'ler sağa kaydırılır.

a=0xC9 >> 1 örneğini inceleyelim.



Yeni değer a=0x64 olur.

4.7.5. Unary Operatörler

C programlama dilinde diğer dillerde bulunmayan unary operatörler tanımlanmıştır. İki tane unary operatör vardır. Bunlar **++** ve **--** dir. **++** operatörü, değişkenin tuttuğu sayıya 1 ekler. **–** operatörü ise, değişkenin tuttuğu sayıdan 1 çıkarır.

x++ → x=x+1 anlamındadır.

x-- → x=x-1 anlamındadır.

Unary operatörün iki kullanım şekli vardır. ++ ve -- operatörleri değişkenin önüne veya sonrasına koyulabilir. Aşağıda bu kullanımlar açıklanmıştır.

```
x=10;
y=x++;
```

yazılırsa. y değişkeninin değeri y=10 olur. Çünkü program ilk başta x değerini y değişkenine aktarır daha sonra x değerinin 1 artırır.

```
x=10;
y=++x;
```

yazılırsa y değişkeninin değeri y=11 olur. Çünkü program ilk başta x değerini 1 artırır daha sonra y değişkenine aktarır.

Not: Aynı öncelik sırasına sahip operatörlerde program soldan sağa doğru sırayla işlem yapar.

4.7.6. Özel Amaçlı Operatörler

Özel amaçlı operatörler aşağıda verildiği gibi iki başlıkta toplanabilir.

4.7.6.1. Atama (=) Operatörü

Örneklerimizde sıkça kullandığımız atama (=) operatörü sağ taraf değerini sol taraftaki değişkene atar. Örneğin;

a = 10; veya S1 = 100; gibi

4.7.6.2. İşlemli Atama Operatörü

İşlemli atama operatörleri belirtilen bir işlemi daha kısa olarak ifade etmemizi sağlar. İşlemli atama operatörleri aşağıda verilmiştir.

= += -= *= /= %= &= ^= |= <<= >>=

Aşağıda işlemli atama operatörlerine örnekler verilmiştir.

```
a += 2;      a = a + 2;
b -= 10;     b = b - 10;
c *= 10;     c = c * 10
```

4.8. CCS C'DE ÖZEL KARAKTERLERİN GÖSTERİMİ

Derleyicide bazı karakterler (#, {, [, vb.) programda komutun içinde kullanılan karakter olarak algılandıkları için program yazarken bu karakterler komut dışında direkt yazılarak kullanılamazlar. Bu tipteki karakterleri kullanabilmek için aynı karakteri çıkaran karşılıklarının yazılması gerekir. Tablo-4.10'da bahsedilen karşılıklar verilmiştir.

Karşılıklar	Özel Karakter
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

Tablo-4.10. *Özel karakterlerin karşılıkları.*

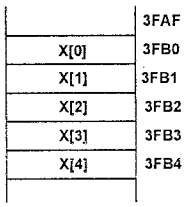
4.9. DİZİ TANIMLAMA

Dizi, aynı tipteki bilgilerin oluşturduğu bütündür. Aynı anda bir değişken ismi altında birden fazla bilgi tutmak amacıyla kullanılır. Diziler aşağıda verilen şekilde tanımlanır ve/veya değer verilir.

<tür> <dizi_ismi> [<Eleman sayısı>];
<tür> <dizi_ismi> [<eleman sayısı>] = { değer, değer,...};

```
char x[5];
```

Örnekte karakter tipinde 5 elemanlı bir dizi tanımlanmıştır. Dizinin ilk elemanı sıfırdan başlar (0), bu nedenle dizinin ilk elemanı x[0], son elemanı ise x[4]'dur. Bu olayı Şekil-4.1'de daha iyi anlayabiliriz. Şekilde kutuların sağındaki numaralar dizi elemanlarının bellekte yerleştirildiği adreslerdir.



Şekil-4.1. *Tek boyutlu dizinin bellekte yerleşimi.*

4.9.1. Dizilere Değer Verme ve Birden Fazla Boyutlu Diziler

Tanımlanan dizilerin elemanlarına değer girilmesi aşağıda verilmiştir.

<tür> <dizi_ismi> [<eleman sayısı>] = { değer, değer, değer, ...};

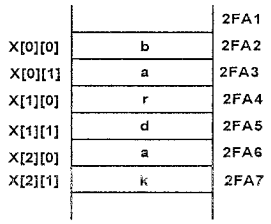
```
int say [10] = {1,5,6,8,9,10,25,4,33,87};
```

Yukarıda tam sayı tipinde 10 elemanlı "say" isminde bir dizi tanımlanmış ve dizi elemanlarına değer verilmiştir. Char tipindeki dizilere değer vermek için aşağıdaki örneği inceleyin.

```
char isim [5] = {'h', 'a', 's', 'a', 'n'};
char isim [ ] = {"hasan"};  // yukarıdaki örnekle aynı anlamdadır
```

Bu örnekte ise karakter tipinde "isim" isminde 5 elemanlı bir dizi tanımlanmıştır. Dikkat edilirse karakter değerleri diziye yüklenirken tek tırnak (' ') içinde belirtilmiştir. Dizi elemanı tanımlarında eleman sayısı boş bırakılabilir. Bu durumda kaç tane elemana değer verilmişse dizi otomatik olarak o kadar elemanlı olur.

Diziler birden fazla boyutluda olabilir. Örneğin aşağıda 2 boyutlu bir dizi ve bellekte yerleşimi görülmektedir.




```
int x [3] [2]={"bardak"};
```

Şekil-4.2. *İki boyutlu dizinin bellek yerleşimi.*

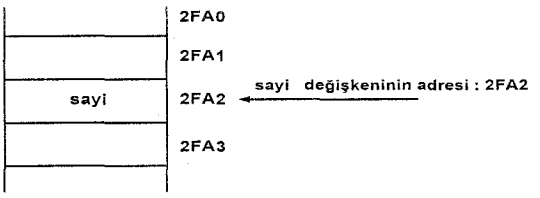
4.10. GÖSTERİCİLER (POINTERS)

Her değişken bilgisayar belleğinde belli bir adreste saklanır. Değişkenlerin adreslerini tutan değişkenlere **gösterici (pointer)** denir. Gösterici kavramı bazı kaynaklarda **işaretci** olarak ta tanımlanmaktadır. Gösterici de bir değişkendir. Göstericiler programcıya bellek adreslerine doğrudan erişim sağladığından, çok daha hızlı, esnek ve verimli programlar yazılmasına olanak sağlar.

Örneğin tamsayı (int) türünde sayi isminde bir değişken tanımlayalım.

```
int sayi;
```

"sayi" değişkeni bilgisayar belleğinde bir yerde saklanır. Farz edelim "sayi" değişkeninin bellekteki adresi 0x2FA2 (heksadesimal olarak) olsun. Bilgisayar belleğini incelersek "sayi" değişkeninin aşağıdaki gibi belleğe yerleşmiş olduğunu görürüz.



Şekil-4.3. *Değişkenin belleğe yerleşmesi.*

İşte göstericiler herhangi bir değişkenin adres bilgilerini saklamak ve adreslerle ilgili işlemler yapmak için kullanılan nesnelerdir. Örnekte verilen "sayi" değişkeninin bellekteki adresini tutan bir değişken tanımladığımızda bu değişken bir gösterici olmuş olur.

4.10.1. Gösterici Bildirimleri

Gösterici bildirimlerinin genel biçimi şöyledir.

<tür> * <gösterici ismi>;

<tür>, göstericinin tuttuğu adreste bulunan değişkenin türüdür. Yani gösterici int türünde bir değişkenin adresini tutuyorsa bu gösterici tanımlanırken int türünde tanımlanmalıdır. Tür char, int, float, vb. gibi herhangi bir tür olabilir. Burada * , işaretçiyi veya adresi temsil etmektedir. Örnek işaretçi bildirimleri aşağıda verilmiştir.

```
int *x ;   // Tamsayı tipinde x adında işaretçi tanımlanıyor.
char *sayi; // Karakter tipinde sayi isminde işaretçi tanımlanıyor.
float *top; // Ondalıklı tipte top isminde işaretçi tanımlanıyor.
unsigned int *ort; // İşaretsiz tamsayı tipinde işaretçi tanımlanıyor.
```

Gösterici bildirimlerinin diğer türlere ilişkin bildirimlerden * atomu ile ayrıldığına dikkat ediniz.

```
char sayi;
```

bildiriminde "sayi" karakter türünden bir değişkendir. Oysa;

```
char *sayi;
```

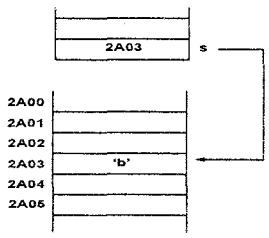
bildiriminde "sayi" karakter türünde bir göstericidir. İçerisinde char tipindeki "sayi" değişkeninin bellekte bulunduğu adres vardır. Bu bildirimden derleyici "sayi" nesnesinin bir gösterici olduğunu, "sayi" nesnesinin içinde bulunan adresin gösterdiği yerin char yani karakter olarak yorumlanacağını anlar.

4.10.2. Gösterici Kullanımı

Aşağıdaki örneği inceleyelim:

```
char *s;
...
s = (char *) 0x2A03;
*s = 'b';
```

Bu örnekte **s** char türünde (tipinde) bir gösterici olarak tanımlanmıştır. Ardından s göstericisi bellekte 0x2A03 adresine yerleştirilmiştir. Komuttaki ***s**, göstericinin gösterdiği adreste bulunan nesnedir. Dolayısıyla yukarıdaki örnekte 'b' karakteri 0x2A03 adresine yazılır (0x o sayının heksadesimal olduğunu belirtir). Şekil-4.4'te bu olay daha net anlaşılabilmektedir.



Şekil-4.4. *Gösterici kullanımı örneği.*

Şimdi de farklı bir örnek olarak aşağıdakini verelim:

```
char *s;
s = (char *) 0x3FB0;
........
*s = 'b';
```

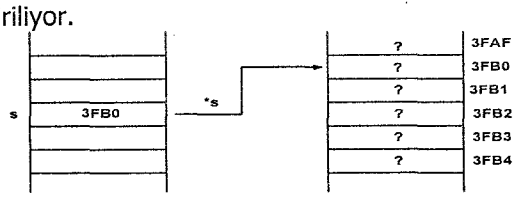
```
++s;
*s = 'c';
.......
```

Bu verdiğimiz örneği örneği adım adım inceleyelim:

1.Adım:

```
char *s;
s = (char *) 0x3FB0;
```

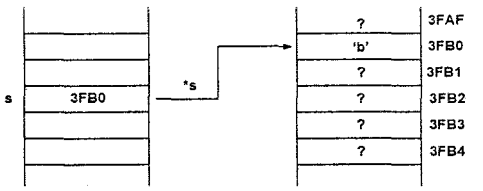
ile karakter türünden **s** isimli gösterici tanımlanmış ve içerisine karakteri gösteren **3FB0** adresi yerleştiriliyor.



Şekil-4.5. *1. adım.*

2.Adım:

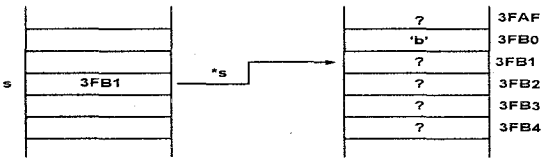
*s = 'b'; komutu ile 0x3FB0 adresine 'b' yazılmıştır.



Şekil-4.6. *2. adım.*

3.Adım:

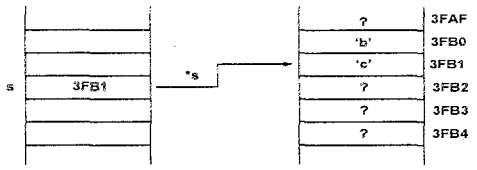
++s; komutu ile s göstericisinin içerisindeki adres (gösterdiği adres) 1 arttırılmıştır. Şimdi ***s, 0x3FB1** adresindeki nesneyi temsil ediyor.



Şekil-4.7. *3. adım.*

4. Adım:

*s = 'c'; komutu ile **0x3FB1** adresine 'c' karakteri yazılıyor.



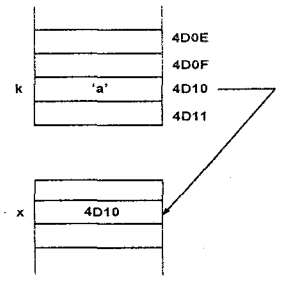
Şekil-4.8. *4. adım.*

4.10.3. Adres Operatörü

& operatörü nesnenin bellekteki adresini öğrenmemizi sağlayan bir operatördür. Örneğin;

```
char  k;        // char tipinde k isminde değişken tanımlanıyor.
char  *x;       // x adında gösterici tanımlanıyor.
.........
k = 'a';        // k değişkeninin içine a karakteri atanıyor.
x = &k;         // k değişkenin adresi x işaretçisine atanıyor.
```

burada **k** değişkeninin adresi x işaretçisine atanmaktadır. Şekil-4.9'de bu daha iyi anlaşılmaktadır (Şekilde k değişkeninin 4D10 adresinde olduğu farz edilmiştir).



Şekil-4.9. *Adres operatörü işlemi.*

4.11. YAPILAR (STRUCTURES)

Yapılar C programlarımızda bize kendi türlerimizi tanıma olanağı sunar. Yapıları programlarınızda kullanarak çok nitelikli kodlar yazabilirsiniz. Örneğin bir rehber programı yapacağımızı düşünelim. Bunun için programımızda kişilerin ad, soyad, telefon no, adres bilgilerini tutacak değişkenler oluşturmamız gerekir.

```
char ad;
char soyad;
int tel;
unsigned  char adres;
```

Normal de oluşturacağımız rehber programı kaç kişilik ise her kişi için yukarıdaki değişkenleri ayrı ayrı tanımlamamız gerekir. Bundan kurtulmak için dizi mantığını kullanabilirsiniz. Bu değişkenleri dizi olarak tanımladığımızı farz edelim. Örneğin her bir dizi 50 üyeli olsun. Rehberimiz 50 kişilik olsun. Biz programda 50 kişilik üye ismi girmesek bile program bellekte her bir dizi için 50 birim yer kaplayacaktır. Böylece kullanmadığımız dizi üyelerine bellekte boşa yer ayırmış olacağız. **Hâlbuki bu işi yapı mantığını kullanarak yaparsak kullanılmayan bir üye için bellekte yer kaplanmaz ve programın anlaşılması ile esnekliği artar. Ayrıca diziler sadece aynı tür nesneleri içinde barındırabilir. Ama yapılar farklı türden nesneleri içinde barındırabilir. Bu özellik yapılar ile dizileri birbirinden ayıran en önemli özelliktir.** Yapı bildirimin genel şekli şöyledir:

```
<struct > [yapı ismi]
{
        <tür> <k1>;
        <tür> <k2>;
        <tür> <k3>;
        ...
};
```

Yukarıda ki örneği bir yapı kullanarak tanımlayalım:

```
struct rehber
{
        char ad[30];
        char soyad[30];
        int tel[7];
        unsigned  char adres[100];
}
```

Artık "rehber" adında bir yapımız var. Bu yapı bu hali ile bellekte yer kaplamaz. Çünkü bu hali ile biz derleyiciye sadece yeni bir tür hakkında bilgi vermekteyiz. Ne zaman biz bir yapı tanımlamasını kullanırız o zaman bellekte yer kaplanır.

Şimdi tanımladığımız yapı türünden bir değişken tanımlayalım.

```
struct rehber onur_bilgi;
```

Yukarıda verilen yapı tanımı ile "onur_bilgi" adında yeni bir değişkenimiz olmuştur. Bu değişkenin türü "rehber" dir ve tanımlandığı için bellekte yer kaplar.

Tanımlanmış bir yapı elemanına erişmek için nokta (.) operatörü kullanılır. Aşağıda ki komut ile rehber yapısında tanımlanmış onur_bilgi değişkeninin ad elemanına erişilmiş ve değer yüklenmiştir.

```
onur_bilgi.ad="Mehmet";
```

4.12. CCS C'DE FONKSİYON TANIMLAMA

C'de alt programlara fonksiyon denir. Her fonksiyon bir takım işlemleri yerine getirmek için tasarlanır ve gerektiğinde çağrılır. Fonksiyonların, onları çağıran fonksiyonlardan aldıkları **girdileri** ve yine onları çağıran fonksiyonlara gönderdikleri **çıktıları (geri dönüş değeri)** vardır. Fonksiyonların girdilerine **parametre ya da argüman** denir. Aşağıda bir fonksiyonun tanımlanması görülmektedir.

```
[Geri Dönüş Değerinin Türü]   <Fonksiyon İsmi>   (tür parametre1, tür parametre 2,...)
{
        ..............................................
        ..............................................
}
```

Örnek;

```
int  sicaklik_oku( int x, float y)
{
        .............
        return i;
}
```

Yukarıdaki örnekte *sicaklik_oku* isimli bir fonksiyon tanımlanmıştır. Tanımlanan fonksiyonların parametreleri (girdileri) x ve y'dir, çıktısı ise z'dir. Parametrelerin (x,y) türü float yani ondalıklı sayı olarak tanımlanmış, geri dönüş değerinin (z) türü ise tam sayı olarak belirlenmiştir. Fonksiyonun geri dönüş değeri belirtilmeyebilir. Bu durumda geri dönüş değeri türü kısmına hiç bir şey yazılmaz veya **void** yazılır.
NOT: C'de fonksiyon içinde fonksiyon tanımlanamadığı unutulmamalıdır.

CCS C'de fonksiyon çağırma operatörü olarak **( )** kullanılmaktadır. Bir fonksiyon çağrıldığında programın akışı çağrılan fonksiyona gider, fonksiyonun işlemlerini yapar ve kaldığı yerden ana programa devam eder. Aşağıda önceden tanımlanmış *sicaklik_oku* adlı fonksiyonun çağrılması görülmektedir. Fonksiyon ilk tanıtıldığında kaç tane parametreden oluşuyorsa, fonksiyon çağrılırken o kadar parametre değeri parantez içine yazılmalıdır.

```
sicaklik_oku(20, 5);
```

4.13. AÇIKLAYICI YAZIMI

Açıklayıcı yazılar derleyici tarafından dikkate alınmayan fakat programın anlaşılmasında programı yazana ve programı inceleyen başka kişilere kolay anlama sağlaması açısından yazılan bilgi yazılarıdır. İki tane eğik çizgiden "//" sonra yazılanlar program tarafından derleme işleminde dikkate alınmazlar.

```
output_high(pin_B0);   // B0 pin'i lojik 1 yapılıyor.
```

Bu örnekte "B0 pin'i lojik 1 yapılıyor." ifadesi dikkate alınmaz.

Aynı zamanda **"/*" karakteri ile "*/"** karakteri arasında kalan tüm yazılarda program tarafından dikkate alınmaz.

```
/* Bu program ile belirlenen analog girişlerden gelen sinyali
dijital bilgiye çevirir. Elde edilen dijital bilgi LCD'ye yansıtılır. */
```

4.14. CCS C PROGRAMI GENEL GÖVDESİ

CCS C'de programlarken yazılırken kullanılan belli bir sıra vardır. Program mantığına ters düşmediği sürece istenen tasarımda komutlar programda yer alabilir. Fakat programın okunabilirliğini ve anlaşılmasını arttırmak ve hata yapma olasılığını en aza indirmek için aşağıda verilen program gövdesi kullanılmalıdır.

```
// Program hakkında tanıtıcı bilgiler
/*****************************************************
Program İsmi:
Programın Amacı:
Kullanılan Mikrodenetleyici:
Yazan  :
*****************************************************/
#include <16f877.h>
#include <input.c>
#include <lcd.c>
#fuses HS,NOWDT,NOPROTECT,...
#use delay (clock=20000000)

int x,y;
char bilgi;
long int data;
const int dizi[]={0x23, 0x34};

#int_donanım kesmesi ismi
void fonksiyon ismi ()
{
    Komut veya komutlar;
}
```

Başlık (Header) ve kütüphane dosyalarının tanıtılması, konfigürasyon tanımlamalarının belirtilmesi.

Genel (Global) değişkenler ile sabitlerin tanımlanması

Varsa kullanılan donanım kesmesi fonksiyon tanımlamaları

```
void fonksiyon ismi ()
{
    Komut veya komutlar;
}
```

Varsa kullanıcı tarafından tanımlanan fonksiyonlar (alt programlar)

```
void main ( )
{
    Komut veya
    komutlar;
}
```

Ana program fonksiyonu

Örnek program;

```
/*****************************************************************
            TANITIM AMAÇLI ÖRNEK PROGRAM
Programda kullanılan komutlar tanıtım amaçlıdır.
Yazan    :SERDAR ÇİÇEK
*****************************************************************/
#include <16f876.h> // Kullanılacak denetleyicinin başlık (header)
                    // dosyası tanıtılıyor.
#include <LCD.c>

#fuses HS,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT
#use delay (clock=20000000) // Gecikme fonksiyonda kullanılacak
                            // osilatör frekansı

#int_timer0   // Timer 0 taşma kesmesi tanımlanıyor
void timer_0 kesmesi
{
   set_timer0(15);
   output_high(pin_D0);
   output_low(pin_D0);
}

void matris_veri_yaz (int veri[],int sure)  // Kullanıcı tarafından
{                                           // oluşturulan fonksiyon
   int i,j;

   for (i=0;i<sure;i++)
      {
         for(j=0;j<=4;j++)
         {
            output_c(sutuntara(j));
            output_b(veri[j]);
            delay_ms(10);
         }
      }
}
```

```
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   set_tris_b(0x00);       // B port'u tümüyle çıkış olarak ayarlandı.
   output_high(pin_B0);     // B0 pin'i lojik 1 yapılıyor.
   delay_ms(500);           // 500 ms'lik bir gecikme veriliyor.
   matris_veri_yaz(a,30);   // Oluşturulan fonksiyon çağrılıyor
}
//*****************************************************************
```

4.15. CCS KOMUT SETİ

CCS komutları (fonksiyonları) ileriki bölümlerde uygulama kısımlarında adım adım incelenmiştir. CCS komutlarından **Ön İşlemci Fonksiyonları** Tablo-4.11'de, **Yerleşik Fonksiyonlar (Built-in Functions)** ise Tablo-4.12'de, verilmiştir. Ön işlemci fonksiyonları bu bölümde incelenmiştir. Yerleşik fonksiyonlar ise diğer bölümlerde kısım kısım uygulama örneklerindeki devre programları üzerinde incelenmiştir.

Ön İşlemci (Pre-Processor) Fonksiyonları

Standart C	Yerleşik Kütüphane Fonksiyonları	Derleyici Kontrol Fonksiyonları
#define	#use delay clock	#case
#else	#use fast_io	#opt
#elif	#use fixed_io	#priority
#endif	#use i2c	#list
#error	#use rs232	#nolist
#if	#use standard_io	#org
#ifdef	**Derleyici Fonksiyonları**	#ignore_warnings
#include	_data_	**Hafıza Kontrol Fonk.**
#pragma	_device_	#asm
#undef	_file_	#bit
Denetleyici Tanımlama Fonksiyonları	_line_	#build
#device chip	_pcb_	#byte
#fuses	_pcm_	#endasm
#id	_pch_	#fill_rom
#id checksum	_time_	#locate
#id number		#reserve
#serialize		#rom
#type		#zero_ram

Tablo-4.11. *CCS C Ön işlemci fonksiyonları.*

Yerleşik Fonksiyonlar (Built-in Functions)

Standart C Karakter Fonksiyonları	Standart C Hafıza Fonksiyonları	Standart C Matamatik Fonksiyonları
atoi()	memset()	abs()
atol()	memcpy()	abs32()
atoi32()	calloc()	acos()
atof()	free()	asin()
tolower()	longjmp()	atan()
toupper()	malloc()	ceil()
isalnum()	memchr()	cos()
isalpha()	memcmp()	exp()
isamoung()	memmove()	floor()
isdigit()	offsetof()	labs()
islower()	offsetofbit()	log()
isspace()	realloc()	log10()
isupper()	setjmp()	pwr()
isxdigit()	**RS232 Fonksiyonları**	sin()
strlen()	putc()	sqrt()
strcpy()	putchar()	tan()
strncpy()	gets()	**A/D Fonksiyonları**
strcopy()	puts()	setup_adc_ports()
strcmp()	printf()	setup_adc()
stricmp()	kbhit()	set_adc_channel()
strncmp()	fgetc()	read_adc()
strcat()	fgets()	**Analog Karşılaştırıcı Fonksiyonları**
strstr()	fprintf()	setup_comparator()
strchr()	perror()	**Voltaj Referans Fonksiyonları**
strrchr()	assert()	setup_vref()
strtok()	setup_uart()	setup_low_volt_detect()
strspn()	set_uart_speed()	**Dahili EEPROM Fonksiyonları**
strcspn()	**I2C Fonksiyonları**	read_eeprom()
strpbrk()	i2c_start()	write_eeprom()
strlwr()	i2c_stop()	read_program_eeprom()
Gecikme Fonksiyonları	i2c_read()	write_program_eeprom()
delay_cycles()	i2c_write()	read_calibration()
delay_us()	i2c_poll()	**Bit İşleme Fonksiyonları**
delay_ms()	i2c_isr_state()	shift_right()
CCP Fonksiyonları	**Giriş/Çıkış Fonksiyonları**	shift_left()
setup_ccpX()	output_low()	rotate_right()

set_pwmX_duty()	output_high()	rotate_left()
setup_power_pwm()	output_float()	bit_clear()
setup_power_pwm_pins()	output_bit()	bit_set()
setup_power_pwmx_duty()	output_toggle()	bit_test()
setup_power_pwm_override()	output_X()	swap()
CPU Kontrol Fonk.	input_state()	make8()
sleep()	input()	make16()
reset_cpu()	input_X()	make32()
restart_cause()	set_trix_X()	**Dahili LCD Donanımı Fonksiyonları**
disable_interrupts()	get_tris_x()	LCD_load()
enable_interrupts()	port_a_pullups()	LCD_symbol()
ext_int_edge()	port_b_pullups()	setup_LCD()
read_bank()	**SPI Fonksiyonları**	**PSP Fonksiyonları**
getenv()	setup_spi()	setup_psp()
label_address()	spi_read()	psp_input_full()
goto_address()	spi_write()	psp_output_full()
setup_oscillator()	spi_data_is_in()	psp_overflow()
jump_to_isr()	spi_xfer()	**Zamanlayıcı (Timer) Fonksiyonları**
		setup_timer_X()
		set_timer_X()
		get_timer_X()
		setup_counters()
		setup_wdt()
		restart_wdt()

Tablo-4.12. *CCS C Yerleşik fonksiyonları.*

4.16. ÖN İŞLEMCİ FONKSİYONLARI

Bu bölümde CCS C derleyicisinde kullanılan ön işlemci komutlarından bazılarının açıklamasını yapacağız. Genelde programlama yapılırken en çok kullanılan ön işlemci komutları üzerinde durulmuştur. Anlatılan komutlar uygulama örnekleri kısmında daha iyi anlaşılabilir.

#ASM
#ENDASM

Bu komutlarla #ASM ile #ENDASM arasına programda assembly komutları yazılabilir. Böylece C ile programlama yapılırken istenen yerde assembly komutları kullanılabilir.

```
#asm
        movlw  0x90
        movwf  count
#endasm
```

DATE
Bu işlemci komutu kullanıldığında derlenen dosyanın derlenme tarihini verir.

```
printf("Programın Derlenme Tarihi ");
printf(__DATE__);
```

Yukarıdaki komutlarla ekrana programın derlenme tarihi otomatik olarak yansıtılır.

#DEFINE
Bu komut ile sabit tanımlaması yapılır.

```
#define pi 3.14     // Bu komut ile program her pi gördüğü yerde 3.14
                    // varmış gibi işlem yapar.
x=pi*4;Komutu       // x=3.14*4 anlamın da işlem görür.
```

#DEVICE
Programda kullanılan PIC mikrodenetleyisini ve özelliklerini belirtmek için kullanılır.

```
#device PIC16F876
#device PIC16F877  *=16  ADC=10
```

Bu komutla PIC16F877 denetleyicisinin kullanıldığını, 16 bit'lik işaretçi kullanılacağını (*=16), kullanılacak ADC'nin 10 bit'lik olacağını belirtiyor (ADC=10).

#FUSES
Bu komut ile kullanılan mikrodenetleyici ile ilgili konfigürasyon bit'lerinin durumu belirtilir

#fuses *konfigürasyon kodları,...*

```
#fuses HS,NOWDT
```

Bu komut bize devrede yüksek hızlı kristal osilatör kullanıldığını ve Watchdog Timer'ın kapalı olacağını belirtir. Aşağıda diğer konfigürasyon kodları ve açılımları verilmiştir.

LP = **L**ow **P**ower Osc. - Düşük frekanslı kristal.
XT = Kristal veya seramik resonatör (**Xt**al)
HS = **H**igh **S**peed - Yüksek hızlı kristal veya seramik resonatör
RC = **R**esistor **C**apacitor - Direnç-kondansatör osilatör
WDT = **W**atch **D**og **T**imer kullanılmak istenirse seçilir.
NOWDT = **N**o **W**atch **D**og **T**imer. WDT kullanılmayacaksa seçilir.
PUT = **P**ower **U**p **T**imer isteniyorsa seçilir
NOPUT = **N**o **P**ower **U**p **T**imer. PUT istenmiyorsa seçilir.
LVP = **L**ow **V**oltage **P**rogramming. Düşük gerilimle programlama modu seçilir. PIC'i programlamak için normalde MCLR ucundan

13V gerilim gerekmektedir. LVP modu seçilirse bu gerilim değerinin 5V olması yeterlidir.

NOLVP = **N**o **L**ow **V**oltage **P**rogramming. LVP modu istenmediğinde seçilir.

CPD = **C**ode **P**rotected **D**ata. Data EEPROM belleğinin korumalı olması istendiğinde seçilir.

NOCPD = **N**o **C**ode **P**rotected **D**ata. EEPROM belleği koruması istenmediğinde kullanılır.

PROTECT = Program belleğine (ROM) yazılan verilerin okunmasını engeller.

PROTECT_5% = ROM belleğin %5'lik kısmını okumaya karşı korur.

PROTECT_50% = ROM belleğin %50'lik kısmını okumaya karşı korur.

NOPROTECT = ROM belleğe okumaya karşı koruma koymaz.

#INCLUDE

Bu komut ile kullanılacak mikrodenetleyicinin tanımlama dosyası derleyiciye tanıtılır.

```
#include <16F876.h>
```

#INT_XXXX

Bu komut ile kesmeler (interrupts) tanımlanır. Kesme fonksiyonları aşağıda verilmiştir.

#INT_EEPROM → EEPROM yazma işlemi bitince meydana gelen kesme
#INT_EXT → Dış kesme
#INT_EXT1 → Dış kesme 1
#INT_EXT2 → Dış kesme 2
#INT_I2C → I2C Kesmesi
#INT_LOWVOLT → Düşük voltaj tespit kesmesi
#INT_RB → B Port'unun B4-B7 pin'lerinde herhangi bir değişiklik olduğunda meydana gelen kesme
#INT_RC → C Port'unun C4-C7 pin'lerinde herhangi bir değişiklik olduğunda meydana gelen kesme
#INT_RDA → RS232 data alma kesmesi
#INT_RTCC → Timer0 kesmesi
#INT_SSP → SPI veya I2C faaliyette kesmesi
#INT_TBE RS232 → gönderme tamponu boş kesmesi
#INT_TIMER0 → Timer 0 kesmesi
#INT_TIMER1 → Timer 1 kesmesi
#INT_TIMER2 → Timer 2 kesmesi
#INT_TIMER3 → Timer 3 kesmesi

#PRIORITY

Denetleyici kendine yüklenen programı icra ederken bir kesme meydana geldiğinde o kesmeyi icra eder. Fakat aynı anda iki kesme meydana geldiğinde öncelikli olan

kesmeyi icra eder. İşte bu komut hangi kesmenin daha öncelikli olacağını derleyici programa bildirmeye yarar.

```
#priority  ext, timer0, EEPROM
```

Yukarıdaki örnekte en yüksek öncelikli olan kesmenin int_ext kesmesi, ondan sonra öncelikli olan kesmenin int_timer0 kesmesi ve en düşük önceliğe sahip kesmenin ise int_EEPROM kesmesi olduğu derleyiciye bildirilmektedir. Öncelik belirlenirken yazılan kesme isimlerinin önüne int_ ön ekinin yazılmadığına dikkat ediniz.

TIME

Bu işlemci komutu kullanıldığında derlenen dosyanın derlenme saatini ssat:dakika:saniye formatında verir.

```
printf("Programın Derlenme Saati ");
printf(__TIME__);
```

Yukarıdaki komutlarla ekrana programın derlenme saati otomatik olarak yansıtılır.

#IF, #ELSE, #ELIF, #ENDIF

Programda tanımlanmış değerler üzerinden istenen bir şartın sağlanıp sağlanmadığını, şart sağlanıyorsa ve/veya sağlanmıyorsa hangi komutların işleneceğini belirten bir ön işlemci kontrol komutudur. Kontrol döngüsü #if ile başlar #endif ile sonlanır.

```
#if  limit> 255                 // eğer limit değeri 255'den büyükse
        unsigned int deger;     // bu komutu işle
#else                           // eğer limit değeri 255'den büyük değilse
        int deger;              // bu komutu işle
#endif                          // sonlandır
```

Yukarıdaki örnekte programda tanımlanmış "limit" değişkeninin değeri 255'den büyükse "deger" değişkeni *unsigned int* türünden, eğer limit değişkeni değeri 255'ten büyük değilse "deger" değişkeni *int* türünde tanımlanacaktır.

#IFDEF, #IFNDEF, #ELSE, #ELIF, #ENDIF

Bu komutun bir önce anlatılan #if komutlarında farkı, programda bir C programı şartı değil de, ön işlemci tanımlaması yapılıp yapılmadığını yapıldıysa ve/veya yapılmadıysa programda istenen komutları işletmeye yarayan bir fonksiyondur.

```
#ifdef __pcm__
        #device PIC16C71
#else
        #device PIC16F877
#endif
```

Yukarıda verilen örnekte eğer programda _PCM_ komutu tanıtılmışsa, derleyici hemen altında bulunan komutu işleyip direkt #endif'e atlayacak ve kontrol döngüsünden çıkacaktır. Eğer _PCM_ komutu programda tanıtılmamışsa #else kısmının altındaki komut işlem görecek ve döngü #endif'e gelerek sonlanacaktır.

#USE DELAY

Program yazarken "delay" gecikme komutunun kullanılabilmesi için bu ön işlemci komutunun kullanılması gerekir. Bu komutun kullanımı;

```
#use delay (clock=4000000)
```
şeklindedir.

Bu komutla derleyiciye 4 Mhz'lik (4000000hz) bir osilatör frekansında çalıştığımız bildirilmektedir. Clock ifadesi eşitliğine kullanılan osilatör frekansı hz (hertz) olarak yazılmalıdır.

#DEVICE

Bu komut ile belirtilen PIC denetleyicisi ile ilgili bazı denetleyici ayarları derleyiciye tanıtılır.

```
#device PIC16F877            // PIC16F877 kullanıldığı belirtiliyor
#device PIC16F877 adc=10     // denetleyicide kullanılacak ADC'nin 10 bit
                                olacağı belirtiliyor.
```

#ORG

Bu komut ile istenen komutların ROM bellekte istenen adresten başlanarak yerleştirilmesine veya komutların istenen ROM bellek adres sınırları içinde yerleştirilmesini sağlar.

```
#org    0x1E02, 0x1FFF
void deneme ()
{ ...
}
```

Yukarıdaki örnekte deneme fonksiyonu komutları ROM bellekte 1E02 adresinden başlanarak en fazla 1FFF adresine kadar olan alana yerleştirilecek.

```
#org 0x1C00
bilgi_oku() { }
```

Yukarıdaki örnekte ise bilgi_oku fonksiyonu ROM bellekte 1C00 adresinden itibaren yerleştirilmeye başlanacak.

#ROM

Bu komut ile ROM bellekte istenen bir adres ve/veya adreslere istenen bir bilgi yazılır.

```
#rom 0x2000={1,2,3,4,5}
```

Örnekte verilen komut ile ROM bellekte 2000 adresinden başlanarak sırayla adreslere parantez içinde verilen değerler yerleştirilmektedir.

#RESERVE

Bu komut, RAM bölgesinde istenen bir adres veya adreslerin veya istenen adres aralığının bir amaç için ayrılması gereken zamanlarda kullanılır. Ayrılan adresler derleyici tarafından kullanılmayacaktır. Bu komut #device komutundan sonra kullanılmalıdır.

```
#device PIC16F877

#reserve 0x65              // 0x65 adresi derleyici tarafından kullanılamaz
#reserve 0x53, 0x60, 0x5A  // belirtilen adresler ayrılmıştır
#reserve 0x61:0x71         // belirtilen adres aralığı ayrılmıştır ve derleyici
                              tarafından kullanılamaz.
```

#BYTE

Bu komut ile belirtilen isimde bir değişken veya sabit 1 byte'lık yani 8 bit'lik bir adrese yerleştirilir.

```
#byte  porta = 5
#byte  portb = 6
```

Yukarıdaki örnekte porta ve portb değişkenleri sırayla 5 ve 6 adreslerine yerleştirilmiştir. Yani PIC'in RAM belleğinde bulunan A ve B port adreslerine bu komutlar sayesinde direkt olarak porta ve portb değişkenleri ile erişilebilmektedir.

NOT: Bu komut ile değişken bellekte istenen adrese yerleştirilmektedir. Fakat derleyici otomatik değişken yerleştirmelerinde başka bir değişkeni #byte komutu ile belirlediğiniz aynı adrese yerleştirebilir. Bunu önlemek için #LOCATE komutu kullanılabilir.
Yukarıdaki örnekte 0x05 ve 0x06 adresleri özel amaçlı kaydedici (SFR) olan porta ve portb adresleri olduğu için derleyici bu adreslere otomatik olarak başka bir değişken atamaz.

#LOCATE

Bu komut #byte komutu ile aynı görevi yapar. Yani istenen bir değişkeni istenen bir adrese yerleştirir. #byte komutundan farklı olarak da, kullanılan adresin derleyici programdan tarafından başka amaçlar için kullanılmasını önler.

```
#locate x=0x56
```

#BIT
Bu komut ile istenen bir değişken veya sabit, istenen bir adresin istenen bit'i ile iliştirilebilir.

```
int durum;
#bit okuma = durum.2
okuma=1; // durum değişkeninin değerinin 2.bit'i 1 yapılıyor.
```

Örnekte "okuma" adlı değişken, "durum" değişkeninin 2. bit'ini ifade edecek şekilde tanımlanmaktadır. #bit ile tanımlanan değişken sadece 0 ve 1 değerini alabilir.

#ZERO_RAM
Bu komut ile dâhili kaydedicilerin içerikleri sıfır (0) bilgisi ile sıfırlanır.

#USE STANDART_IO()
PIC mikrodenetleyicilerinde hangi port'un hangi pin'lerinin giriş ve/veya çıkış olarak kullanılacağının programda bildirilmesi gereklidir. CCS C derleyicisinde bu iş için komut mevcuttur. **Fakat derleyicide bu iş için başka komut kullanılmamışsa #use standart_io () komutu standart olarak kullanılmış sayılır.** Bu ön işlemci komutu kullanıldığında programda hangi port'un hangi bit'inin giriş ve/veya çıkış olacağının bildirilmesine gerek yoktur. Bir giriş komutu kullanıldığında derleyici ilgili pin'i giriş olarak yönlendirir. Bir çıkış komutu kullanıldığında ise derleyici ilgili pin'i çıkış olarak yönlendirir. Bu komut bir sonraki #use xxx_io komutuna kadar geçerliliğini korur.

#use standart_io (*port ismi*) *Örnek: #use standart_io(a)*

#USE FAST_IO()
Bu komut kullanıldığında **programda mutlaka set_tris_X() komutu ile hangi port'un hangi pin'inin giriş veya çıkış olarak kullanılacağının bildirilmesi gereklidir.** Komutta parantez içine hangi port kullanılacaksa o port yazılmalıdır. Bu komut bir sonraki #use xxx_io komutuna kadar geçerliliğini korur.

#use fast_io(*port ismi*)

```
#use fast_io(a)
set_tris_a(0xFF);      // A port'u komple giriş yapılıyor.
```

4.17. MİKRODENETLEYİCİ KONTROL FONKSİYONLARI

SLEEP( )
Bu komut ile mikrodenetleyici uyku (sleep) moduna geçer. Kullanımı;

sleep( ); şeklindedir.

Sleep modundan denetleyiciyi çıkarmak için; aşağıdaki yollardan biri uygulanır.

$\overline{\text{MCLR}}$ reseti
WDT reseti
INT ucundan gelen kesmeler, RB Port'unda meydana gelen değişim kesmesi ve çeşitli çevresel kesmeler (Timer1 kesmesi, EEPROM yazma işlemi tamamlandı kesmesi, USART RX, TX kesmeleri, R/C osilatör kaynaklı AD çevriminde, CCP yakalama modu kesmesi gibi).

RESET_CPU( )
Kullanılan mikrodenetleyicinin resetlenmesini sağlar. Kullanımı;

reset_cpu( ); şeklindedir.

RESTART_CAUSE( )
Reset olan mikrodenetleyicinin en son hangi olay nedeniyle kendini sıfırladığını (resetlediğini) öğrenmemizi sağlayan komuttur. Bu fonksiyon kullanıldığında belli anahtar kelimeler ile geri döner. Bunlardan bazıları şunlardır; WDT_FROM_SLEEP, WDT_TIMEOUT, MCLR_FROM_SLEEP, NORMAL_POWER UP.

***değişken ismi* = restart_cause( );**

ENABLE_INTERRUPTS( )
Bu komut belirtilen kesmeyi aktif hale getirmeye yarar. Kesme isimler .h uzantısına sahip denetleyici başlık dosyalarında görülebilir (16f877.h gibi).

enable_interrupts(*kesme ismi*);

```
enable_interrupts ( int_timer0 );     // Timer0 kesmesi aktif hale gelir.
enable_interrupts ( int_timer1 );     // Timer1 kesmesi aktif hale gelir.
enable_interrupts (global);          // Program içinde belirtilen (aktif hale
                                        getirilen) tüm kesmeleri kullanabilmek
                                        için bu komutu kullanmamız gereklidir.
```

DISABLE_INTERRUPTS( )
Bu komut belirtilen kesmeyi pasif hale getirmeye yarar.

disable_interrupts(*kesme ismi*);

```
disable_interrupts (int_timer0);   // Timer0 kesmesi pasif hale gelir.
disable_interrupts (INT_RDA);      // RS232 kesmesi pasif hale gelir.
disable_interrupts (global);       // Tüm kesmeleri pasif hale getir demektir.
```

CLEAR_INTERRUPT( )

Bir kesme meydana geldiğinde o kesmeye ait kesme bayrağı aktif olur. Bayrağın aktif olması istenen kesmenin meydana geldiğini belirtir. Bu fonksiyon ile istenen kesme bayrağı sıfırlanır.

clear_interrupt(*kesme ismi*);

```
clear_interrupt( timer0 );  // Timer0 kesmesi bayrağını pasif hale getirir
```

READ_BANK( )

Bu komut ile istenen bellek bank'ındaki (bank0, bank1, bank2, bank3 - PIC çeşidine göre değişir) kullanıcı RAM alanından istenen bilgi okunabilir. Komuttaki bank kısmına istenen bellek bankını (0, 1, 2, 3), ofset kısmına da okunacak adres girilmelidir.

***değişken ismi* = read_bank (*bank, ofset*);**

```
x = read_bank(1, 12);  //  Bank1 RAM bölgesinde 0x12 adresindeki bilgi
                           okunuyor.
```

WRITE_BANK( )

İstenen bellek bank'ındaki (bank0, bank1, bank2, bank3 - PIC çeşidine göre değişir) kullanıcı RAM alanının istenen yerine bilgi yazmaya yarar.

write_bank(*bank, ofset, yazılacak değer*);

```
int8 x;
x=0x59;
write_bank(1, 12, x);  // Bank1 RAM alanında 0x12 adresine 0x59 değeri
yazılıyor.
```

GOTO_ADDRESS( )

ROM alanında istenen adrese dallanmayı (gitmeyi) sağlar. Gidilecek adres bilgisi 16 veya 32 bit olmalıdır.

goto_address(*gidilecek adres*);

```
const   int16   x = 0x2814;   // 16 bit'lik x adında bir sabit tanımlanıyor.
goto_address( x );            // Program 0x2814 adresine gider.
```

LABEL_ADDRESS( )
Bu komut istenen etiketten sonra gelecek ilk komutun adresini öğrenmeye yarar.

***değişken* ismi = label_address(*etiket ismi*);**

```
oku:
a=read_bank(1,19);
...
b=label_address(oku);  // oku etiketinden sonraki ilk komutun adresini alır
                          b değişkenine yükler.
```

GETENV( )
Bu komut ile bilinen anahtar kelimeleri yazarak program ve derleyici ile ilgili bilgiler alınır.

***değişken ismi* = getenv("*anahtar kelime*");**

```
if getenv("version")<3.235
        printf(lcd_putc, "Program versiyonu eski");
```

Yukarıdaki örnekte "version" anahtar kelimesi kullanılarak programın versiyonu derleyiciden alınmaktadır. Bunun gibi isteğe uygun anahtar kelimeler kullanarak istenen bilgiler derleyiciden alınabilir. Diğer bazı kullanılabilecek anahtar kelimeler aşağıda verilmiştir. Eğer kullanılan anahtar kelime derleyici tarafından tanınmansa komut "0" ile geri döner.

STACK = Stack boyutunu verir.
DATA_EEPROM = DATA EEPROM belleğinin byte olarak büyüklüğünü verir.
ADC_CHANNELS = A/D çevirici kanal sayısını verir.
READ_PROGRAM = Kod belleği okunabiliyorsa 1 değeri ile geri döner.
COMP = Kullanılan mikrodenetleyicide karşılaştırıcı varsa 1 değeri ile geri döner.
ID = Kullanılan mikrodenetleyici ID'si ile geri döner.
DEVICE = Kullanılan mikrodenetleyici ismi ile geri döner.
VERSION = Kullanılan derleyici sürümü bilgisi ile geri döner.
CAN = Kullanılan mikrodenetleyicide CAN varsa 1 değeri ile geri döner.
SPI = Kullanılan mikrodenetleyicide SPI varsa 1 değeri ile geri döner.
TIMERx = Kullanılan mikrodenetleyicide timerx (timer0, timer1 vb.) varsa 1 değeri ile geri döner.
FUSE_VALID: konfigürasyon özellik ismi = Eğer yazılan konfigürasyon özelliği (BROWNOUT, WDT, PUT) varsa 1 değeri ile geri döner.

EXT_INT_EDGE( )
Bu komut ile dış kesmenin (EXT) dıştan gelen sinyalin düşen mi yoksa yükselen kenarında mı aktif olacağının seçimi yapılır.

ext_int_edge(*kaynak, sinyalin istenen aktif kenarı*);

Komutun *kaynak* kısmına PIC18XXX modelleri için 0, 1 veya 2 (EXT0, EXT1, EXT2 anlamında) yazılır, diğer PIC modelleri için komutun *kaynak* kısmı yazılmaz. Komutta *sinyalin aktif kenarı* kısmına aşağıdaki tanımlamalar yazılır;

Yükselen kenar için = L_TO_H
Düşen kenar için = H_TO_L

```
ext_int_edge( 2, L_TO_H );// PIC18XXX denetleyici için EXT2 dış kesmesi ayarı.
ext_int_edge( H_TO_L );   // Diğer tüm modeller için EXT dış kesmesi ayarı.
```

4.18. KARAKTER FONKSİYONLARI

Standart C ile ilgili karakter işlem fonksiyonları CCS C derleyicisi içinde geçerlidir. Aşağıda bu fonksiyonların bazılarının kısaca kullanım amaçları verilmiştir.

atoi(), atol(), atoi32()
Bu fonksiyonlar string ifadeleri tamsayı ifadelere çevirir. Aralarındaki fark çevirdikleri tamsayı ifadelerin bit uzunluğudur. Eğer tamsayıya çevrilecek string ifade değeri 8 bit'lik sınırlar içindeyse atoi(), 16 bit'lik sayı sınırları içindeyse atol(), çevrilecek sayı 16 bit'lik sayı sınırından büyükse atoi32() komutu kullanılır. **Bu fonksiyonların kullanılması için programın başında stdlib.h dosyasının tanıtılması gereklidir (#include <stdlib.h> şeklinde).**

***8 bit'lik değişken ismi* = atoi(*string ifade*);**
***16 bit'lik değişken ismi* = atol(*string ifade*);**
***32 bit'lik değişken ismi* = atoi32(*string ifade*);**

```
char data[ ]={"123"};
int i;
i=atoi(data);   // i değişkeni değeri tamsayı olarak 123 olur.
```

atof()
Bu komutta string ifadeyi tamsayıya çevirir. Fakat bu komut ile yapılan çevirim sonucu 32 bit'lik ondalıklı sayıdır. Bu fonksiyonun kullanılması için programın başında **stdlib.h** dosyasının tanıtılması gereklidir (#include <stdlib.h> şeklinde).

***32 bit'lik ondalıklı tipte değişken* = atof (*string ifade*);**

```
char data[ ]={"123.456"};
float k;
k=atof(data);   // k değişkeni değeri ondalıklı olarak 123.456 olur.
```

ISALNUM( ), ISALPHA( ), ISDIGIT( ), ISLOWER( ), ISSPACE( ), ISUPPER( ), ISXDIGIT( ), ISCNTRL( ), ISGRAPH( ), ISPRINT( ), ISPUNCT( )

Bu fonksiyonlar belirtilen değerlerin istenen kritere uyup uymadıklarını kontrol ederler. Eğer belirtilen değer karşılaştırılan kritere uyuyorsa fonksiyonlar TRUE (1) değeri ile uymuyorsa FALSE (0) değeri ile geri dönerler. Fonksiyonlardaki *"datac"* kısmına 8 bit'lik karakterler yazılabilir. **Bu fonksiyonun kullanılabilmesi için ctype.h dosyasının programın başında tanıtılması gerekmektedir (#include <ctype.h> şeklinde).**

***değişken ismi* = isalnum(*datac*);**
***değişken ismi* = isalpha(*datac*);**
***değişken ismi* = isdigit(*datac*);**
***değişken ismi* = islower(*datac*);**
***değişken ismi* = isspace(*datac*);**
***değişken ismi* = isupper(*datac*);**
***değişken ismi* = isxdigit(*datac*);**
iscntrl(*x*);
isgraph(*x*);
isprint(*x*);
ispunct(*x*);

Aşağıda yukarıda verilen fonksiyonların hangi durumu test ettiği açıklanarak verilmiştir.

isalnum(x) = X yerine yazılan karakterin 0...9, A...Z veya a...z karakterlerinden biri olduğunun doğruluğunu kontrol eder.
isalpha(x) = X yerine yazılan karakterin A...Z veya a...z karakterlerinden biri olduğunun doğruluğunu kontrol eder.
isdigit(x) = X yerine yazılan karakterin 0...9 karakterlerinden biri olduğunun doğruluğunu kontrol eder.
islower(x) = X yerine yazılan karakterin a...z karakterlerinden biri olduğunun doğruluğunu kontrol eder.
isupper(x) = X yerine yazılan karakterin A...Z karakterlerinden biri olduğunun doğruluğunu kontrol eder.
isspace(x) = X'in boşluk (space) karakteri olduğunun doğruluğunu kontrol eder.
isxdigit(x) = X yerine yazılan karakterin 0...9, A...F veya a...f karakterlerinden biri olduğunun doğruluğunu kontrol eder.
iscntrl(x) = X yerine yazılan karakter boşluk karakterinden küçük olduğunun doğruluğunu kontrol eder.
isgraph(x) = X yerine yazılan karakter boşluk karakterinden büyük olduğunun doğruluğunu kontrol eder.
isprint(x) = X yerine yazılan karakter boşluk karakterinden büyük veya eşit olduğunun doğruluğunu kontrol eder.

ispunct(x) = X yerine yazılan karakter boşluk karakterinden büyük ve harf ya da karakter olmadığının doğruluğunu test eder.

TOLOWER( ), TOUPPER( )
Bu fonksiyon alfabetik harfleri büyük harften küçük harfe, küçük harften büyük harfe çevirir. Karakterler tek tırnak içinde gösterilir.

değişken ismi = tolower(*değer*); // Harflerin büyük harf karşılığını verir.
değişken ismi = toupper(*değer*); // Harflerin küçük harf karşılığını verir.

x = tolower('D'); // x'in değeri d olur.
x= toupper ('r'); // x'in değeri R olur.

ISAMOUNG( )
Bir karakterin verilen string ifade içinde olup olmadığını test eder. Eğer sonuç doğruysa TRUE (1) değeri ile yanlış ise FALSE (0) değeri ile geri döner. String ifade çift tırnak içinde gösterilir.

değişken ismi = isamoung(*karakter, string*);

char x;
x = isamoung (x, "ABCDEFGHIJKLMNOPRSTUVYZ");

STRCAT( ), STRCMP( ), STRNCMP( ), STRLEN( ), STRLWR( )
Bu fonksiyonlar string ifadeler ile ilgili bazı işlemlerin yapılmasını sağlarlar. **Bu fonksiyonların kullanılabilmesi için string.h dosyasının programın başında tanıtılması gerekmektedir (#include <string.h> şeklinde).**

dizi ismi = strcat(*S1,S2*); // S1 ve S2 stringlerini birleştirir.
değişken ismi = strcmp (*S1, S2*); // S1 ve S2 stringlerini karşılaştırır.
d*eğişken ismi* = strncmp (*S1, S2,n*); // S1 ve S2 stringlerinin n. Değerini // karşılaştır.
değişken ismi = strlen (*S1, S2*); // S1 stringinin kaç karakter olduğunu // belirtir.
dizi ismi = strlwr(*S1*); // S1 stringinin tüm karakterlerini küçük karaktere // çevirir.

Fonksiyonlardaki *S1, S2* tanımları direkt string ifadeyi değil ("merhaba" gibi), dizi ismini ifade eder (s1[ ]={"veli"}; gibi).

STRCPY( )
Bir string dizisine bir başka string ifadeyi kopyalar.

strcpy(*dizi ismi, string*);

```
char deger[5];
strcpy(deger, "ahmet"); // deger dizisine "ahmet" ifadesi kopyalanmış olur.
```

4.19. MATEMATİK FONKSİYONLARI

Matematik fonksiyonları standart C matematik fonksiyonları ile aynıdır. Aşağıda bu fonksiyonların bazılarının kısaca kullanım amaçları verilmiştir. **Matematik fonksiyonlarını kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.**

ABS( ), FABS( )
Sayının mutlak değerini alır. **Bu fonksiyonları kullanmak için diğer matematik fonksiyonlarından farklı olarak programın başında stdlib.h dosyasının tanıtılması gerekmektedir.** İki fonksiyon arasında fark olmamakla beraber FABS( ) fonksiyonu normalde ondalıklı sayılar için kullanılır.

```
x=abs(-90);  // x'in değeri 90 olur.
x=fabs(-90); // x'in değeri 90 olur.
```

SIN( ), COS( ), TAN( ), ASIN( ), ACOS( ), ATAN( ), SINH( ), COSH( ), TANH( ), ATAN2( )
Trigonometrik işlemlerin yapılmasını sağlayan fonksiyonlardır. **Bu fonksiyonları kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.** math.c dosyasının içinde π değeri (PI) sabit olarak tanımlanmıştır (#define PI 3.141592654 komutu ile). Eğer kullanıcı kendi programında bir PI değeri tanımlayacaksa bu tanımlamayı math.c dosyasını programa tanıtan komuttan önce yapmalıdır.

değişken ismi = sin (*rad*);
değişken ismi = cos (*rad*);
değişken ismi = tan (*rad*);
değişken ismi = asin (*val*);
değişken ismi = acos (*val*);
değişken ismi = atan (*val*);
değişken ismi = atan2 (*val, val*);
değişken ismi = sinh (*value*);
değişken ismi = cosh (*value*);
değişken ismi = tanh (*value*);

Fonksiyonlardaki *"rad"* kısmına -2π ile 2π arasında istenen bir açı değeri radyan cinsinden (ondalıklı olabilir) yazılabilir. Fonksiyonlardaki *"val"* kısmına ise -1.0 ile 1.0 arasında ondalıklı bir değer yazılabilir. Fonksiyonlardaki *"value"* kısmına ondalıklı herhangi bir değer yazılabilir.

sin = İstenen açının sinüs değerini hesaplar.
cos = İstenen açının kosinüs değerini hesaplar.
tan = İstenen açının tanjant değerini hesaplar.
asin = İstenen açının arc sinüs değerini hesaplar.
acos = İstenen açının arc kosinüs değerini hesaplar.
atan = İstenen açının arc tanjant değerini hesaplar.
sinh = İstenen açının hiperbolik sinüs değerini hesaplar.
cosh = İstenen açının hiperbolik cosinüs değerini hesaplar.
tanh = İstenen açının hiperbolik tanjant değerini hesaplar.

atan2 (y,x) = y/x değerinin arc tanjant değerini hesaplar.
x = cos(PI); // x=-1 olur.

LOG10( )
İstenen değerin 10 tabanındaki logaritmasını hesaplar ve sonuçla geri döner. Fonksiyonda değer kısmına tam veya ondalıklı bir değer girilir. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = log10(*değer*);

LOG( )
İstenen değerin doğal logaritmasını hesaplar (ln). Fonksiyonda değer kısmına tam veya ondalıklı bir değer girilir. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = log(*değer*);

SQRT( )
İstenen pozitif bir değerin karekökünü alan matematik fonksiyonudur. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = sqrt(*değer*);

EXP( )
Bu fonksiyon istenen değerin üstel fonksiyonunu hesaplar. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = exp(*değer*);

POW( )
İstenen sayının istenen sayıda üstünü alan bir fonksiyondur. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = pow(*değer, değer*);

x = pow(2, 4); // x'in değeri 16 olur.

FLOOR( )
İstenen değerin tam kısmını alan fonksiyondur. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = floor(*değer*);

x = floor(13.6); // x değeri 13 olur.

CEIL( )
Bu fonksiyon ondalıklı bir değeri bir üst sayıya tamamlar. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = ceil(*değer*);
x = ceil(13.2) ; // x'in değeri 14 olur.
x = ceil(17.9) ; // x'in değeri 18 olur.

FMOD( )
Bu fonksiyon iki sayının birbiri ile bölümünden kalanı verir. Bu fonksiyonu kullanmak için programın başında math.h dosyasının tanıtılması gerekmektedir.

değişken ismi = fmod(*değer, değer*);

x = fmod (26, 3); // x'in değeri 2 olur.

5. Bölüm

CCS C İle Program Derlemek ve Denemek

- CCS C'de Derleme İşlemi
- PIC PROG/DEKA Deneme Kartı
- PIC'e Program Yüklemek (IC-PROG İle)

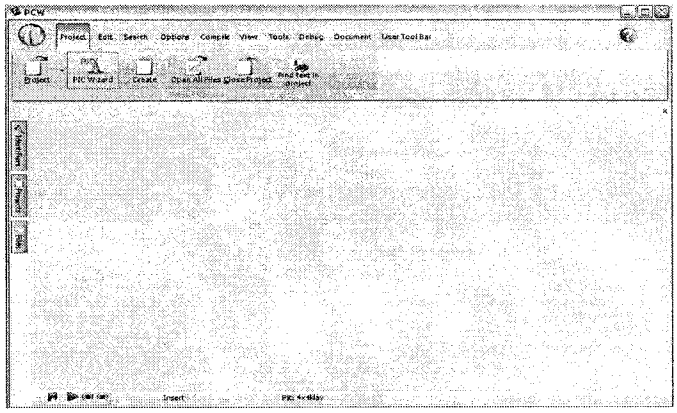
5.1. CCS C'DE DERLEME İŞLEMİ

Bu bölümde CCS C derleyicisi ile bir programın yazılımı ve derleme aşamaları adım adım anlatılmıştır. Derleme işlemi yeni sürüm olan v4.*** sürümünde anlatılmaktadır. Genel olarak derleme ayarları ve derleme işlemi eski sürüm olan v3.*** serisi CCS C derleyicilerinde de aynıdır. Sadece sürüm 4 sürümlerinde küçük farklılıklar ve program ara yüz farkı vardır.

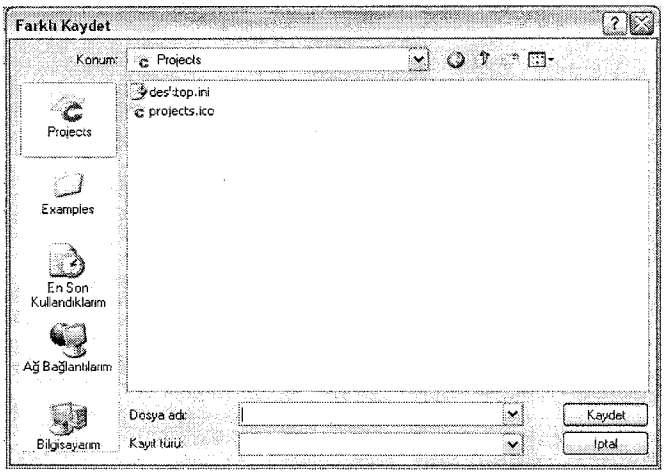
Derleme işlemi için mutlaka bir proje dosyası oluşturulmalıdır. Proje dosyası oluşturmak için 2 yöntem vardır. Bunlar otomatik ve manüel (el ile) proje dosyası oluşturmaktır.

5.1.1. Otomatik Olarak Proje Dosyası Oluşturmak ve Derlemek

Otomatik olarak proje dosyası oluşturmak için ilk başta "Project" menüsü seçilmeli ve menü altında açılan sekmelerden **"PIC Wizard"** sekmesi seçilmelidir (Şekil-5.1). Önümüze gelen menüde proje dosyasının kaydedileceği yer ve ismi belirtilir (Şekil-5.2).

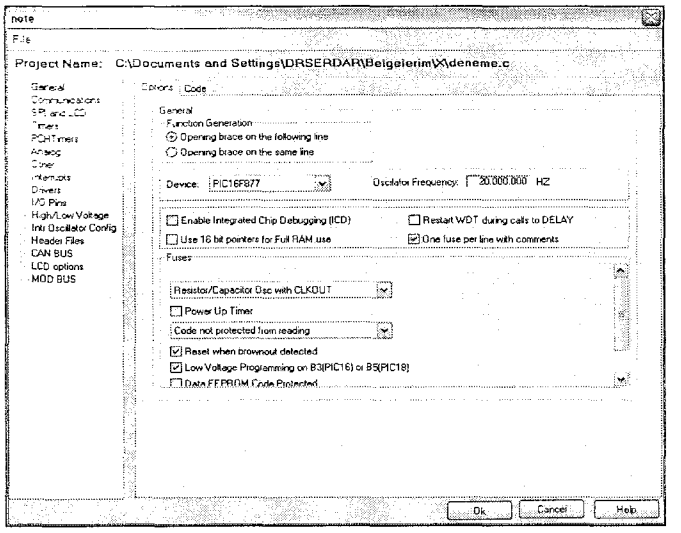


Şekil-5.1. *Otomatik proje dosyası oluşturma penceresi.*



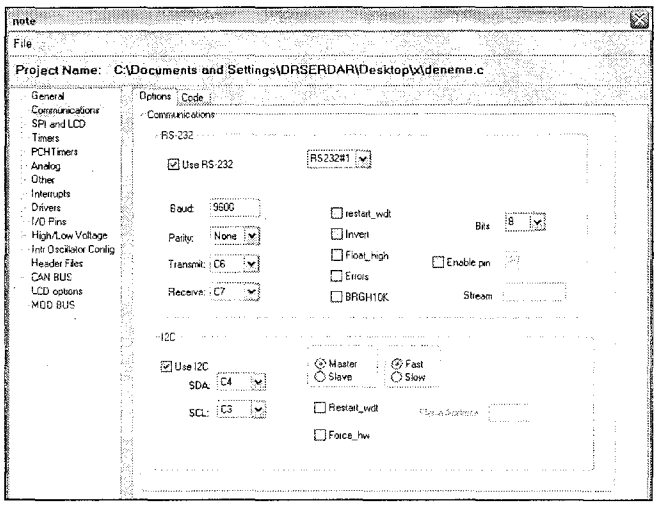
Şekil-5.2. *Proje dosyasının kayıt yeri penceresi.*

Bu işlem de yapıldıktan sonra ekrana proje dosyası ile ilgili ayarlar menüsü (Şekil-5.3) gelir. Ayar menüsünde sol tarafta çeşitli ayar sekmeleri bulunmaktadır. Buradan istenen sekme tıklandığında seçilen sekme ile ilgili seçenekler ekrana gelir. Buralardan programınızda kullanmak istediğiniz donanımlar, kesmeler, port giriş/çıkış durumları, zamanlayıcılar, ADC, seri iletişim, I^2C vb. ile ilgili seçim ve ayarların kod karşılıklarının programınıza otomatik olarak geçmesini sağlarsınız. GENERAL sekmesindeki "Device" kısmında kullanılacak PIC mikrodenetleyicisi, "Oscilator Frequency" kısmında osilatör frekansı, "Fuses" kısmında ise kullanılacak PIC için başlangıç ayarları (kullanılacak osilatör tipi, kod koruma istekleri, istenen reset işlemleri vb.) belirtilir.



Şekil-5.3. *Proje dosyası ayar menüsü.*

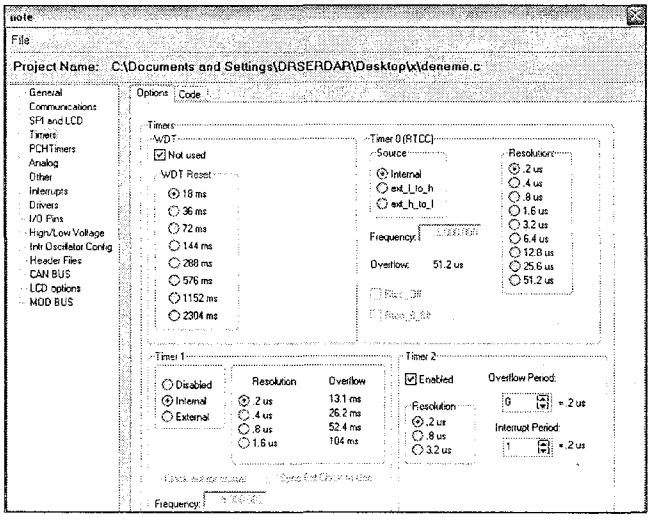
Menünün alt kısmında COMMUNICATIONS sekmesinde ise (Şekil-5.4) projemizde seri iletişim veya I^2C iletişimi kullanılacaksa bunlarla ilgili ayarlar yapılır. Bu kısımda kullanılacak özellik yanına tik işareti konularak aktif hale getirilir ve gerekli ayarlar yapılır.



Şekil-5.4. *Communications sekmesi penceresi.*

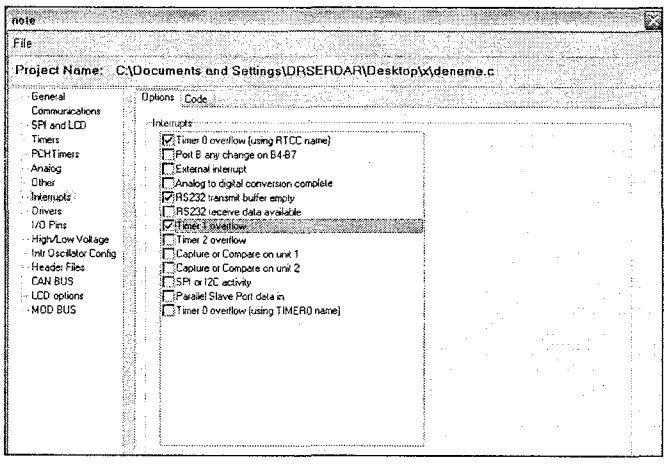
TIMERS sekmesinde ise kullanılmak istenen zamanlayıcılar ile ilgili ayarlar yapılır (Şekil-5.5). Kullanılmak istenen Timer için Enable (açık) veya Disable (kapalı) seçeneklerinden biri seçilerek kullanılacak zamanlayıcı/zamanlayıcılar seçilir. Seçim

yapıldıktan sonra zamanlayıcı ile ilgili ayarlar yapılır. Bu menüdeki "Resolutions" kısmında seçilen değerlere göre istenen kesmenin ne kadar zaman sonra olacağı belirlenir. Seçilen Resolutions değerine göre "Overflow" ve "Overflow Period" kısımlarında taşma süreleri otomatik olarak program tarafından hesaplanır. Böylece kullanıcı istediği taşma süresine (kesme süresine) sahip bir zamanlayıcı ayarı yapmış olur.



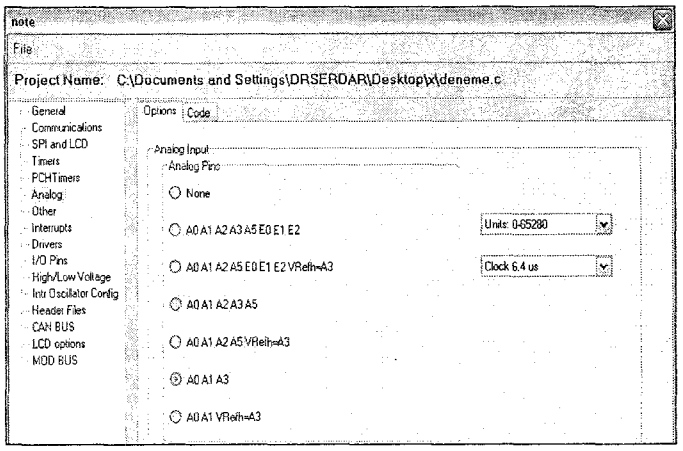
Şekil-5.5. *Timers sekmesi penceresi.*

Örneğin diğer bir sekme olan INTERRUPTS sekmesi kullanılmak istenen kesmelerin seçimini sağlar (Şekil-5.6). Böylece kolayca kesmelerle ilgili fonksiyon programa eklenir. İstenen kesmeye tik işareti (✓) konulur.

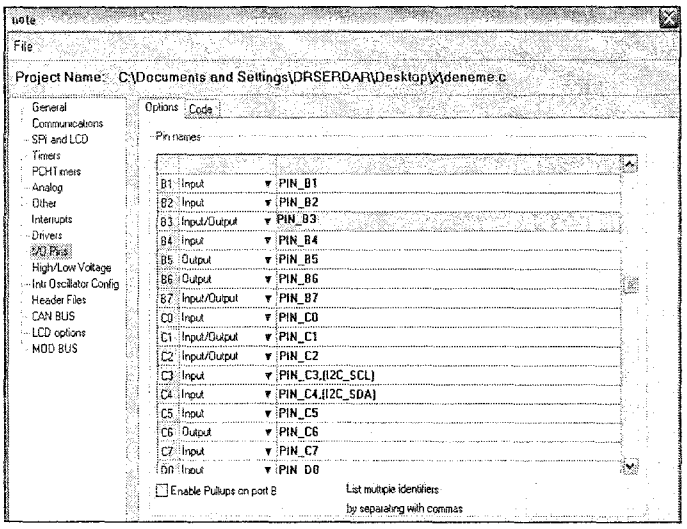


Şekil-5.6. *Interrupts sekmesi penceresi.*

ADC birimini kullanmak isteniyorsa ANALOG sekmesi tıklanır (Şekil-5.7). Bu menüde ise ADC için kullanılacak analog giriş uçları ve V_{REF} ucu seçimi yapılır. Ayrıca ADC biriminin kaç bit'lik olacağı (Units kısmından) ve clok sinyali seçimleri yapılır. Denetleyicinin pin'lerinin hangisinin giriş, hangisinin çıkış, hangi pin'in analog giriş ve hangi pin'in kullanılmayacağı I/O PINS sekmesi seçilerek yapılabilir (Şekil-5.8).

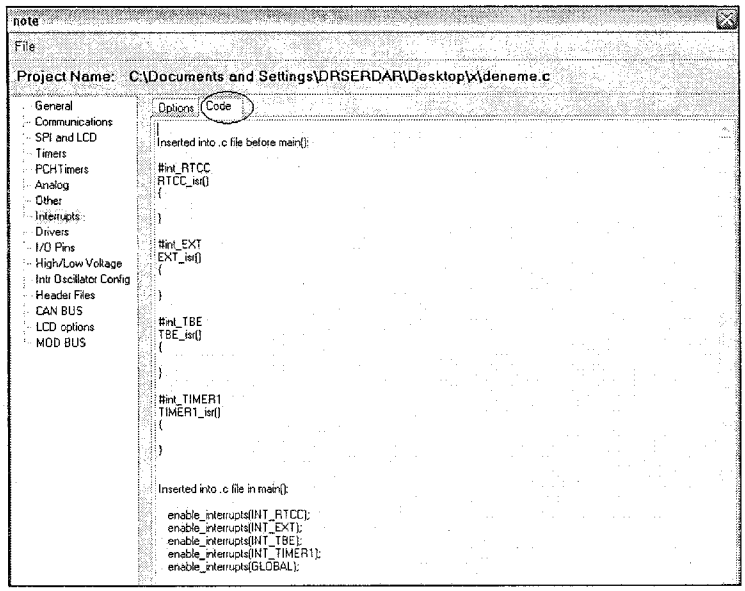


Şekil-5.7. *Analog sekmesi penceresi.*



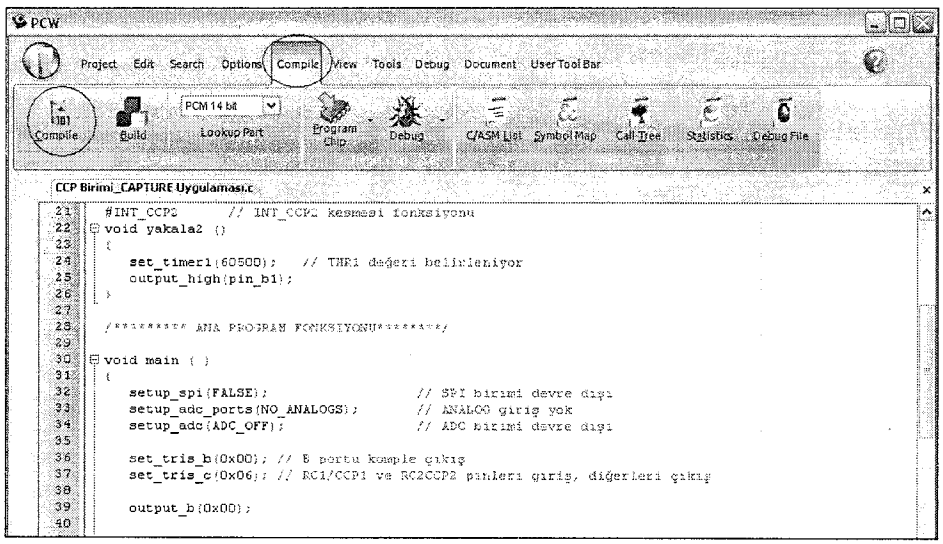
Şekil-5.8. *I/O Pins sekmesi penceresi.*

Yapılan her ayar sırasında "Code" butonu tıklanırsa o sekmede yapılan ayarların kod karşılığı görülür (Şekil-5.9).

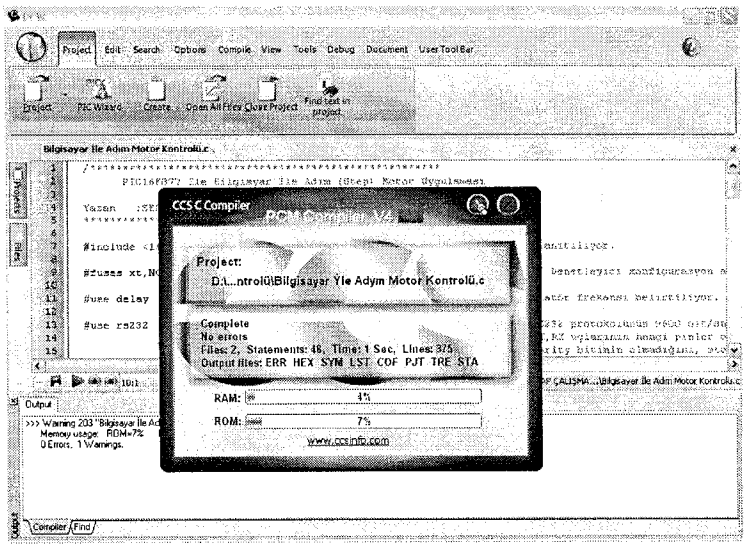


Şekil-5.9. *Otomatik oluşturulan kodların görünmesi.*

Dosyanın derlenmesi için **"Compile"** menüsüne tıklanır ve gelen sekmelerden "Compile" butonuna tıklanır (Şekil-5.10) veya farenin sağ tuşuna basılır ve açılan menüden "Compile" komutu seçilir veya F9 tuşuna basılır. Derleme işlemi sonunda ekrana derleme sonuçlarıyla ilgili bilgiler gelir (Şekil-5.11).



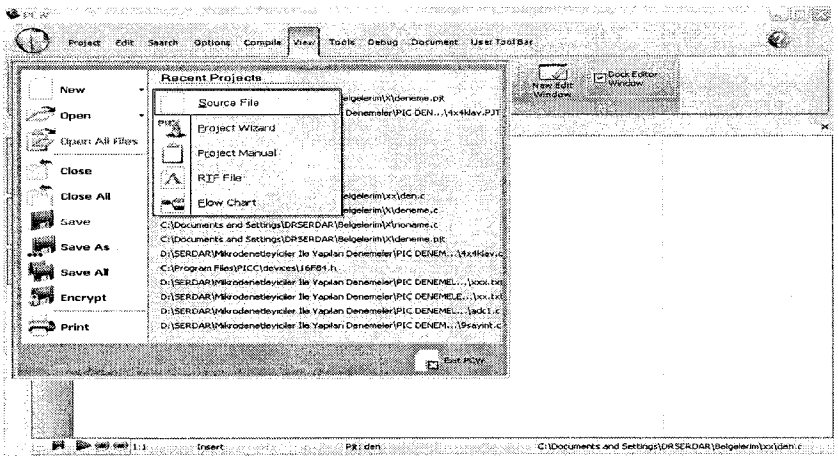
Şekil-5.10. *Oluşturulan bir proje dosyasının derlenmesi.*



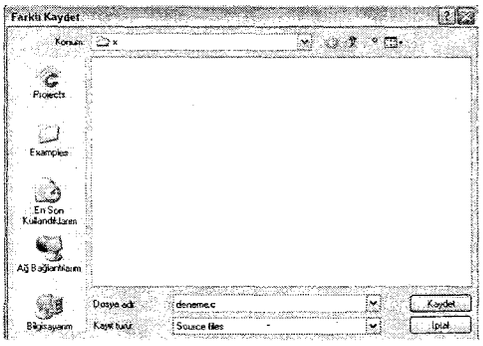
Şekil-5.11. *Derleme işlemi sonucu ekran görüntüsü.*

5.1.2. El İle (Manuel) Proje Dosyası Oluşturmak Ve Derlemek

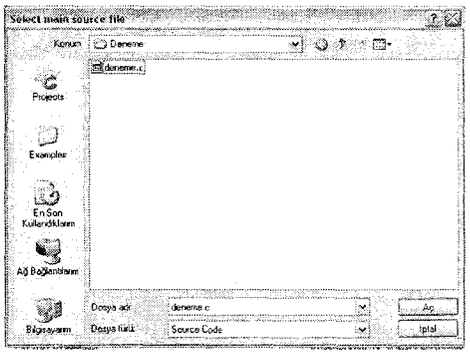
El ile (manuel) bir proje dosyası oluşturmak için ilk başta bir kaynak dosya (Source File) oluşturulmalıdır. Bunun için **"File"** mesünden **"New"** komutu seçilir oradan da **"Source Files"** komutu tıklanır (Şekil-5.12). Gelen menüde dosyanın kaydedileceği yer ve ismi belirtilir (Şekil-5.13). Daha sonra **"Project"** menüsüne tıklanır ve gelen sekmelerden **"Create"** komutu seçilir.



Şekil-5.12. *Kaynak dosya oluşturmak.*

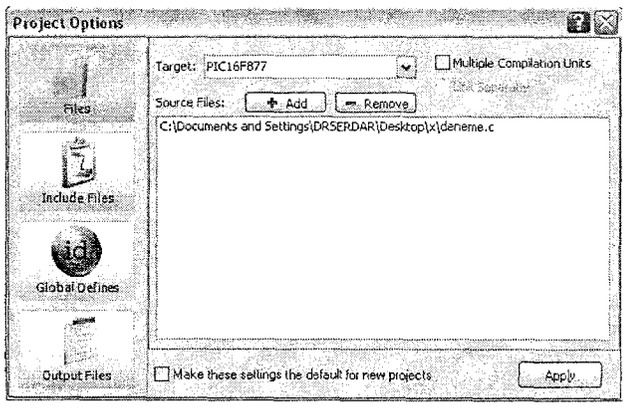


Şekil-5.13. *Kaynak dosyanın kayıt yeri ve isminin belirlenmesi.*



Şekil-5.14. *El ile (Manuel) proje dosyası oluşturmak.*

Önümüze gelen menüde (Şekil-5.14) oluşturulacak proje dosyası için kaynak dosya (.c uzantılı dosya) seçilir (**gelen menüde alt tarafta dosya türü olarak *Source Code* seçilmelidir**). Daha sonra önümüze **"Project Options"** menüsü gelir (Şekil-5.15). Bu menüde kullanılacak mikrodenetleyici ve kullanılacak kaynak dosya (Source Files) seçilir. Bu menüde sol taraftaki sekmelerden **"Include Files"** butonu tıklanarak programın derleme esnasında kullanacağı dosyaların hangi dizinde olduğu belirtilir. **"Output File"** butonuna tıklandığında ise önümüze derleme sonucunda oluşturulacak dosyaların seçildiği menü ekrana gelir. Tüm işlemler yapıldıktan sonra **"Apply"** butonu tıklanır.



Şekil-5.15. *Project Options menüsü.*

Dosyanın derlenmesi için otomatik derleme işlemlerinde anlatıldığı gibi **"Compile"** menüsüne tıklanır ve gelen sekmelerden "Compile" butonuna tıklanır veya farenin sağ tuşuna basılır ve açılan menüden "Compile" komutu seçilir.

5.2. DERLEME SONUCU OLUŞAN VE CCS C'NİN TANIDIĞI DOSYA FORMATLARI

Derleme işlemi sonunda CCS C programı çeşitli uzantıda dosyalar oluşturur. Aşağıda derleme işlemi sonucunda oluşan dosyalar ile CCS C programı dosya formatları ve bu dosyaların ne içerdiği hakkında bilgiler verilmiştir.

.C = C kodlarını barındıran kaynak dosya.
.H = Başlık dosyası veya özel fonksiyon dosyaları.
.PJT = CCS C proje dosyası.
.LST = CCS C komutlarının assembly karşılığının listelendiği dosya.
.SYM = Kaydedicilerin ve program değişkenlerinin kaydedildiği adresleri gösteren dosya.
.STA = RAM, ROM ve Yığın (stack) kullanım istatistiğini veren dosyadır.
.TRE = Programda kullanılan fonksiyonları ve RAM ve ROM'daki adreslerini verir.
.HEX = Programın makine koduna çevrilmiş dosyası.
.COF = Makine kodları ile hata ayıklama (debugger) bilgilerinin olduğu dosya.
.COD = Hata ayıklama bilgilerinin olduğu ikilik kodlardan oluşmuş dosya.
.ERR = Derleme sonucu oluşan hata (error) ve uyarıları (warning) veren dosya.
.ESYM = IDE (Integrated Development Environment - Tümleşik geliştirme ortamı) kullanıcıları için oluşturulan bir dosyadır. Program değişken ve açıklama bilgileri yer alır.
.RTF = CCS C kurulum dizini içinde "rtfeditor.exe" adında bir program vardır. Bu dosya Word programı gibi bir metin editörü programıdır. Bu program ile hazırlanan metin dosyaları .RTF (Rich Text File) uzantılı olarak kaydedilebilir.
.RVF = RVF (Rich View Format) dosya uzantısı, CCS C kurulum dizini içinde "rtfeditor.exe" programı oluşturulan metin dosyalarının asıl kayıt uzantısıdır.
.DGR = CCS C kurulum dizini içinde "flowcharter.exe" adında bir akış diyagramı çizmeye yarayan bir program vardır. Bu program .DGR uzantılı dosyalar oluşturmaktadır.

5.3. PIC PROG/DEKA 16F87X DENEME KARTI

Kitapta incelenen örneklerin bazıları hem proteus benzetim programı hem de PIC PROG/DEKA 16F87X deneme kartı ile denenebilir. Bazı örnekler direkt olarak deneme kartı ile denenemese bile, port çıkışlarına bağlı bulunan precission soket pin'lerinden bir jumper teli ile alınarak deneme kartında bulunmayan başka devrelere bağlanabilir. PIC PROG/DEKA 16F87x, mikrodenetleyicilerin devre üzerinden programlanarak PIC'i sökmeden denenebilmesi amacıyla üretilmiştir. Kart üzerinde 28 ve 40 pin'li PIC'leri seri port'tan programlayabileceğiniz yerleşik JDM programlayıcı mevcuttur. Seri port'u olmayan kullanıcılar için karta ICSP özelliği

eklenmiştir. Bu özellik sayesinde USB port'undan programlama yapan ürünle birlikte PIC'inizi USB port'undan da programlama imkanı yaratabilirsiniz.

5.3.1. PIC PROG/DEKA Deneme Kartı Özellikleri

Deneme kartı üzerinde deneme yapabileceğiniz bir çok çevre birimi ile donatılmıştır. Aşağıda deneme kartının özellikleri maddeler halinde listelenmiştir.

RS232 port'u ile PIC'i devreden çıkarmadan programlama ve deneme imkanı sunar.

Bootloader (açılış yükleyicisi) kullanarak yine RS232 port'undan otomatik programlama ve deneme yapılabilir.

ICSP konnektörü ile ALTAŞ'ın ürünü PIC RPOG .USB/ICSP ve PIC RPOG USB/SERI programlama kartlarına bağlanarak USB port'undan otomatik programlama ve deneme yapılabilir.

28 ve 40 pin'li tüm PIC mikrodenetleyiciler programlanabilir ve birçok fonksiyonu denenebilir.

Bu entegrelerle aşağıda sıralanan özellikler test edilebilir.

A port'u giriş olarak yönlendirilerek butonlarla dijital giriş yapılır. Çıkış olarak yönlendirilerek LED'lere çıkış yapılır.

PORTA I/O pin'leri ile bu port'un tamamını breadboard'a taşıyarak özel projeleriniz için kullanabilirsiniz.

B port'u çıkış port'u olarak düzenlenmiştir. B port'una bağlı 8 led, 2 adet 7 segment display ve LCD display konnektörü bulunmaktadır. Bu üç modülü gereksiniminize göre modül seçme Jumperi ile seçebilirsiniz.

PORTB I/O pin'leri ile B port'unun tamamını breadboard'a taşıyarak da özel projeleriniz için kullanabilirsiniz.

Seri haberleşme (Uzaktan kumanda) için IR (kızıl ötesi) alıcı-verici uygulamaları yapılabilir.

Bilgisayar ile seri haberleşme için MAX232'li haberleşme modülü bulunmaktadır. Bilgisayar ile asenkron seri haberleşme uygulamaları yapılabilir.

24Cxx serisi EEPROM'a yazma ve okuma uygulamaları yapılabilir.

4x4 Tuş takımı uygulamaları yapılabilir.

Buzzer ile ses uygulamaları yapılabilir.

Yine LCD ile ve PORTA butonlarını kullanarak saat projeleri, buzzer'ı da kullanarak alarmlı saat... vb. projeler yapılabilir.

7 Segmet display'leri ve RA5 port'una bağlı AN4 pin'lerine bir PTC bağlamak suretiyle termometre yapabilirsiniz.

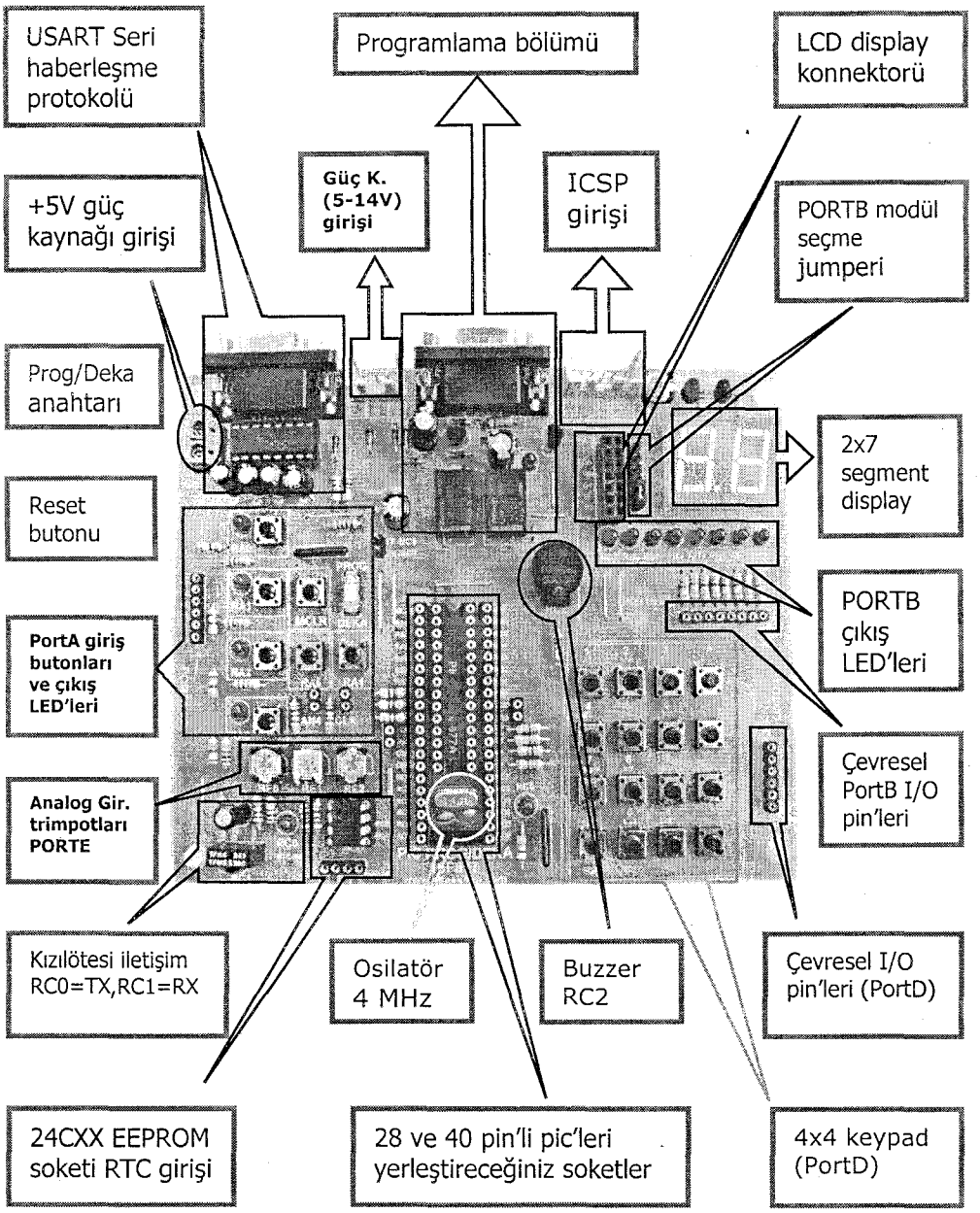
AN4 pin'lerine LDR bağlayarak ışık yoğunluğunu ölçen bir VU metre tasarlayabilirsiniz.

INFRA RED alıcı ile uzaktan aldığınız sinyali USART ile bilgisayara aktarabilirsiniz.

IR LED ve tuş takımını kullanarak 16 butonlu uzaktan kumanda yapabilirsiniz.

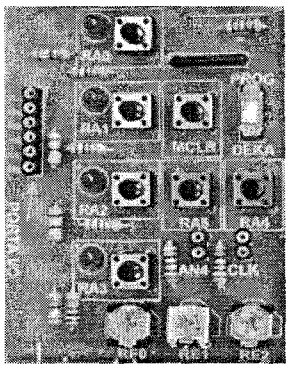
Çıkış pin'lerini breadboard'a taşıyarak kendi özel projelerinizi yapabilirsiniz.

5.3.2. Deneme Kartının Birimleri



Şekil-5.16. *PIC PROG/DEKA 16F87X Deneme kartı.*

5.3.2.1. A Port'u

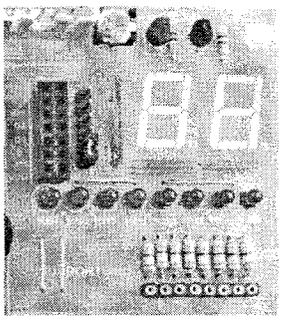


Şekil-5.17. *Port A birimi.*

A port'u genel olarak dijital giriş veya dijital çıkış port'u olarak düzenlenmiştir. A port'undan çıkış yapabilmek için, RA0, RA1, RA2 ve RA3 pin'lerine led yerleştirilmiştir. Bu LED'ler lojik-1'de aktiftir. Yani ilgili port'a 1 (+5V) gönderildiğinde LED yanacaktır. RA0, RA1, RA2 ve RA3 pin'lerinden dijital giriş yapabilmekte mümkündür. Dijital giriş için bu uçlara buton yerleştirilmiştir. Bu butonlar da lojik-1'de aktif tir. Yani bir butona basıldığında yanındaki LED yanar ve ilgili pin'e 1 uygulanır. RA4 ve RA5 pin'lerine ise lojik-0'da aktif butonlar yerleştirilmiştir. RA4'e ve RA5'e aynı zamanda harici giriş yapabilmek için soketler yerleştirilmiştir. Bu soketlerden (RA4) harici clock ya da (RA5) harici analog giriş olarak kullanmak içindir.

5.3.2.2. B Port'u

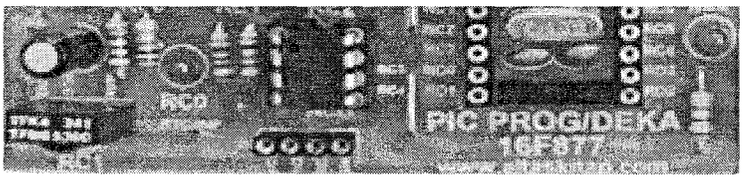
B port'una birbirinden bağımsız 3 modül yerleştirilmiştir. Bunlardan birincisi bar-led (grup led) modülüdür. LED'leri kullanabilmek için modül seçme jumperi (atlama) **LED** konumuna alınmalıdır. BAR-LED uygulamaları için 8 LED lojik-1'de aktif olarak bağlanmıştır. BAR-LED uygulamalarından; binary (00-FF) sayıcı, yürüyen ışık uygulamaları ya da seri haberleşme uygulamalarında 8 bit'lik veri PortB LED'lerinde gösterilebilir. B port'unun ikinci modülü ise 7 segment display'dir. 2 adet ortak katod 7 segment display'in, segment'leri, sırasıyla: RB0 => a, RB1 => b, RB2 => c, RB3 => d, RB4 => e, RB5 =>f, RB6 =>g segment'lerine bağlıdır. Displaylerin katotları ise NPN transistörler ile toprağa bağlanmıştır. Transistörleri A port'unun 0. ve 1. (RA0 ve RA1) bit'leri kontrol etmektedir. Örneğin sağdaki display'de 3 rakamını göstermek için B port'undan binary 00101111 gönderdikten sonra RA1 port'una 1 göndermeniz yeterlidir. İki display'i de kullanabilmeniz için tarama yöntemi kullanmanız gerekmektedir. Bu modülü kullanmak için modül seçme jumper'ını **7seg** konumuna almalısınız.



Şekil-5.18. *Port B birimi.*

5.3.2.3. C Port'u

C port'u genel olarak seri haberleşme için ayrılmıştır. RC0 pin'ine piyasada Infra-red (kızıl ötesi) verici olarak bilinen IR LED yerleştirilmiştir. Gönderilecek dijital veri 30 ile 40 KHz arasında değişen modülasyon frekansı üzerine yüklenerek bu IR LED'den gönderilir. RC1 pin'ine Infra-red (kızıl ötesi) alıcı olarak bilinen (TFMS5300) yerleştirilmiştir. Bu eleman, 38KHz ile modüle edilerek IR LED'den gönderilen dijital verileri, demodüle edecek şekilde düzenlenmiştir. Normalde RC1 pin'ine +5V uygulanır. Modüleli kızıl ötesi ışık TFMS5300'a gönderildiğinde RC0 pin'in gerilim seviyesi 0 Volta düşer. TFMS5300'nin bu özelliği, asenkron seri haberleşme (uzaktan kumanda) için kullanılır.



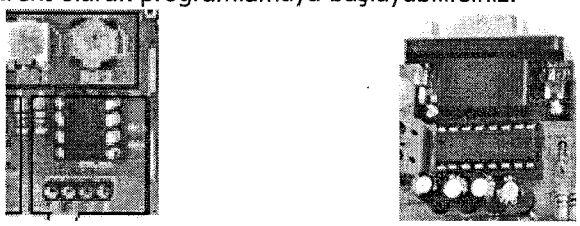
Şekil-5.19. *C Port'u birimleri.*

RC2 pin'i 16F877A'nın donanımsal PWM çıkışıdır. PWM özelliğini kullanabilmeniz için bu uca BUZZER yerleştirilmiştir. Buzzer'den pwm sinyali sayesinde değişik tonda ses çıkarmak mümkündür.

Şekil-5.20. *Deneme kartında bulunan buzzer.*

RC3 ve RC4 port'ları donanımsal olarak I2C EEPROM'a yazma /okuma işlemini gerçekleştirebilmektedir. Hem I2C EEPROM'ların yapısını hem de harici EEPROM

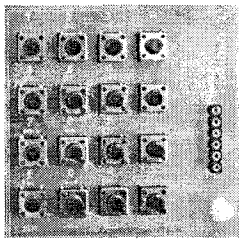
kullanımını öğrenmek için RC3 ve RC4 pin'lerine 24CXX serisi entegreleri takabileceğiniz 8 pin'li soket yerleştirilmiştir. Bu sokete kullanacağınız entegreyi taktıktan sonra direkt olarak programlamaya başlayabilirsiniz.



Şekil-5.21. *I^2C ve USART birimi.*

RC5 pin'inin donanımsal olarak herhangi bir özel fonksiyonu olmadığı (PIC16F87XX) için bu port'a sadece bir LED bağlanmıştır. RC6 ve RC7 donanımsal (USART) asenkron seri haberleşme protokolünü desteklemektedir. Bu yüzden bu port'lar bilgisayar ile seri haberleşme için kullanılacak şekilde düzenlenmiştir. Bilgisayara seri olarak veri göndermek için RC6 (TX) pin'i kullanılır. Bilgisayardan pic'e seri veri göndermek için ise RC7 (RX) pin'i kullanılır. Bu pin'lerin bir diğer özelliği de bootloader'ı desteklemesidir.

5.3.2.4. D Port'u



Şekil-5.22. *D port'una bağlı tuş takımı.*

D portunun pin'lerine, keypad uygulamaları yapabilmek için 4x4=16 butonlu tuş takımı yerleştirilmiştir. Tuş takımları genellikle 3x4=12 butonlu ya da 4x4=16 butonlu olarak imal edilirler. Tuş takımları az sayıda pin kullanarak çok sayıda giriş butonu kontrol etmek için tasarlanmıştır. Örneğin tuş takımından üretilen 16 farklı sayıyı sadece 8 bit'lik D port'u ile okumak mümkündür.

5.3.2.5. E Port'u

E port'u analog giriş yapmak için ayrılmıştır. RE0, RE1, RE2 pin'lerine sırayla 3 adet trimpot yerleştirilmiştir. Trimpot'lar saat yönünde çevrilirse ilgili trimpot'un bağlı olduğu pin'in gerilim seviyesi 0-5V arasında yükselir. Saat yönünün tersine çevrilirse gerilim seviyesi düşer. Bu 0-5V arası gerilim değişikliğini PIC'e okutarak analog-dijital çevirici uygulamaları yapılmaktadır.

5.3.2.6. Deneme Kartı Osilatör Birimi

PIC PROG DEKA 16F87XX piyasada en çok kullanılan 4 MHz kristal osilatör ile çalışmaktadır. PIC'i programlarken osilatör seçeneği olarak **XT** seçilmesi gerekir.

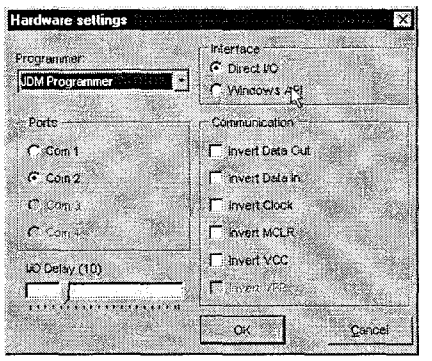
5.4. PIC'E PROGRAM YÜKLEMEK (IC-PROG İLE)

Derleme işlemi sonucunda yazılan programın makine kodu karşılığı olan .hex uzantılı bir dosya oluşur. Bu .hex uzantılı dosyanın denetleyiciye yüklenmesi gerekmektedir. Yükleme işlemi için uygun bir kart ve buna uygun yükleme programı gerekmektedir. **Yükleme işlemi için yukarıda anlatılan PIC PROG/DEKA PIC16F87X deneme kartı da kullanılabilir. Bu kart üzerinde hazır bulunan bir JDM programlayıcı bulunmaktadır.** Bu iş için piyasada en popüler olan JDM programlayıcı kartı ve IC-Prog yazılımıdır. Bu nedenle bu kısımda IC-Prog programından bahsedilmiştir.

IC-Prog programını ilk defa çalıştırdığınızda (icprog.exe) sizden Hardware Settings ekranında donanım ayarlarını yapmanız istenecektir.

İşletim Sisteminiz Win 95/98/ME ise Gerekli Ayarlar

Aşağıdaki ekranda programmer yazan liste kutusundan **JDM Programmer** seçeneğini seçiniz. Kart hangi COM port'a (COM1, COM2...vb.) bağlıysa Ports bölümünde o bağlantı noktasını işaretleyiniz.



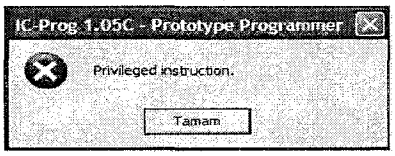
Şekil-5.23. *JDM Hardware ayar penceresi.*

Bilgisayarınızda yüklü olan işletim sistemi Windows 98, 95 ve ME ise Direct I/O seçeneklerini işaretleyiniz. I/O Delay ve Communication bölümlerinde değişiklik yapmanıza gerek yoktur. Ancak farklı anakartların elektriksel özelliklerinden dolayı zamanlama problemleri programlama hatasına neden olabilir. Bu durumda I/O delay kaydırma çubuğunu ayarlayarak kullanılan PIC için en uygun I/O gecikme zamanı seçmeniz gerekebilir. Gerekli konfigurasyonu yaptıktan sonra OK düğmesine tıkladığınızda program çalışır duruma gelecektir. Programı çalıştırdıktan sonra **Settings** bölümünde **Options**'a tıklayarak karşınıza çıkacak penceredeki

language bölümünden dili Turkish seçerek programı **Türkçe olarak** kullanmaya başlayabilirsiniz. Yaptığınız bu ayarların geçerli olması için programı tekrar başlatmanız gerekmektedir.

İşletim Sisteminiz Win XP ise Gerekli Ayarlar

Windows işletim sistemi donanım kaynaklarına doğrudan erişime izin vermediğinden yazılım bu haliyle XP üzerinde programlama yapamaz. Programlama yapmaya çalıştığınızda veya programı yeniden başlattığınızda aşağıdaki hata penceresi ile karşılaşmanız muhtemeldir. Eğer bu durumda programı çalıştırmaya devam ederseniz bilgisayarınızın kilitlenmesine sebep olursunuz.



Şekil-5.24. *XP işletim sisteminde direkt kullanıldığında gelen hata penceresi.*

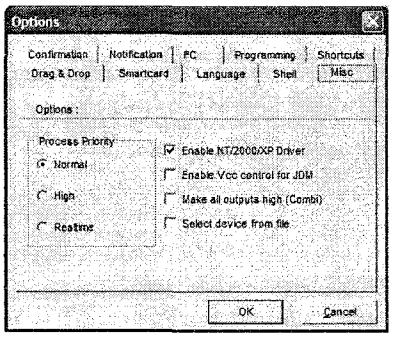
Programın XP ile uyumluluğunu sağlamak için icprog.sys isimli dosyasını icprog.exe dosyasının bulunduğu klasör içerisine kopyalamak gereklidir. IC-Prog programı ve XP için gerekli bu dosya http://www.ic-prog.com/ adresinden indirilebilir. Bu dosyaların tümü aynı zamanda PIC PROG/DEKA PIC16F87XX deneme kartını satın aldığınızda gelen CD içinde de mevcuttur.

Programı tekrar çalıştırdığınızda yukarıdaki hata mesajı ile tekrar karşılaşabilirsiniz. Bu mesajı önemsemeyiniz. Ayarlar (Settings) menüsünden seçenekler (options) seçiniz.

Karşınıza Seçenekler (Options) penceresi geldiğinde Karışık (Misc) sekmesini seçiniz.

Enable NT/2000/XP driver' ı etkin duruma getiriniz.

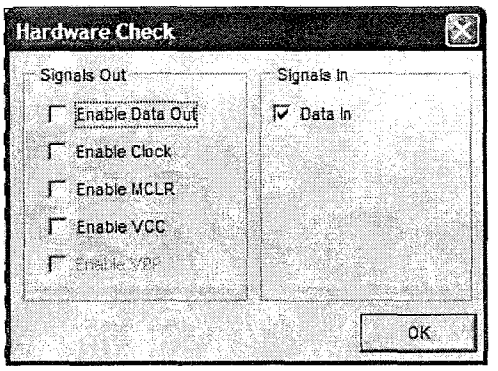
OK düğmesine bastığınızda programınız XP işletim sisteminde sorunsuz programlama yapabilecek duruma gelecektir.



Şekil-5.25. *XP ayarı için options menüsü.*

Donanım Kontrolünün Yapılması

Programlayıcınızı çalıştırmadan önce donanım testi yapmanız, kartınızla ilgili gerekli kurulumları doğru yapıp yapmadığınızı ve PC'nizle uyumlu olarak çalışıp çalışmayacağını kontrol etmenize yarar. Ayarlar (Settings) menüsünden Donanım kontrolü (Hardware Check)'i seçiniz. Aşağıdaki pencere açılacaktır.



Şekil-5.26. *Donanım kontrolü menüsü.*

MCLR etkin (Enable MCLR)'i etkin yapınca PRG (kırmızı LED) yanacaktır. Bu size kartınızın programlayıcı bölümünün çalıştığını gösterir. Ancak kartın tüm fonksiyonlarının çalıştığı anlamına gelmez. Bu kart üzerinde deneme yapabildiğiniz için giriş çıkış elemanlarının çalışıp çalışmadığını cd rom içindeki programları PIC'e yükleyerek görebilirsiniz ya da devre şemasındaki bağlantıya göre kendiniz program yazmanız gerekir.

Mikrodenetleyicinin Seçilmesi

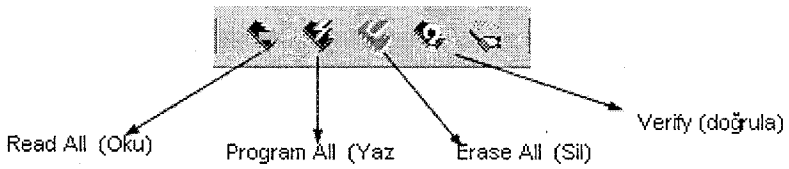
Program penceresinin sağ üst kısmında bulunan liste kutusundan programlamak/denemek istediğiniz entegreyi seçiniz. Aynı işlem Settings (Ayarlar) menüsünde fare işaretçisi Device (Aygıt) seçeneğinin üzerine getirildiğinde çıkan listeden de seçilebilir.

Hex Dosyasının Tampon Belleğe Yerleştirilmesi

Dosya (File) menüsünden dosya aç (Open File)'e tıklayarak karşınıza çıkacak pencerede hex dosyanızı bulup, seçerek hex dosyanızı tampon belleğe yükleyiniz. IC-Prog beş ayrı tampon bellek kullanarak, yani beş ayrı programı hafıza da saklayarak size büyük kolaylık sağlamaktadır. Altta yer alan buffer1, buffer2 sekme butonlarına tıklayarak tampon belleklere ulaşabilirsiniz.

Entegreye Yazma, Yazılanı Okuma, Silme, Doğrulama İşlemlerinin Yapılması

Mikrodenetleyiciye program yazma, mikrodenetleyiciye yazılmış bir programı okuma, program belleğini silme ve program belleğini tampon bellekle karşılaştırma işlemleri icprog programının ana menüsünün üst bölümündeki simgeler üzerine tıklayarak yapılmaktadır. Komut (Command) menüsü altındaki komutlar da aynı işlemleri gerçekleştirmek amacıyla kullanılabilir.



Şekil-5.27. *Komut (Command) menüsü komutları*

Program Belleğini Okuma: Bu simgeye tıkladığınızda karta takılı entegrenin program belleği o an seçili olan tampon belleğe yerleştirilir. Hafızası okunacak entegrenin IC-Prog'dan seçilmiş olması gerekmektedir.

Program Belleğine Yazma: Bu simgeye tıklayarak seçili tampon bellekte bulunan hex dosyası mikrodenetleyicinin hafızasına yüklenir. Karta takılı entegrenin IC-Prog'dan seçilmiş olması gerekmektedir.

Program Belleğini Silme: Bu simgeye tıklayarak karta takılı entegrenin program belleği silinir. Karta takılı entegrenin IC-Prog'dan seçilmiş olması gerekmektedir.

NOT: I2C EEPROM'a yazma işlemleri yapılmak üzere bir EEPROM seçildiğinde silme butonu pasif durumdadır. Ancak bunun pek önemi yoktur. Çünkü EEPROM'a yeni bilgi yazdırıldığında eski veriler elektriksel olarak silinip, yerine yenisi yazılır.

Doğrula: Tampon bellekteki program ile mikrodenetleyicinin belleğindeki program karşılaştırılır.

6. Bölüm

Port Giriş/Çıkış ve Bit/Byte İşlemleri

- set_trix_X()
- get_tris_x()
- output_X()
- output_low()
- output_high()
- output_bit()
- output_toggle()
- output_float()
- input()
- input_X()
- input_state()
- bit_set()
- bit_clear()
- bit_test()
- swap()
- make8()
- make16()
- make32()
- LED Uygulamaları
- Buton Uygulamaları
- port_a_pullups()
- port_b_pullups()

Bu bölümde bir denetleyici uygulamasında bilinmesi gereken temel işlemler olan port ve bit işlemleri incelenmiştir. **Bu kısım aynı zamanda CCS C derleyicisi için ilk adımı ve temel bilgileri içermektedir.** İlk başta port ve bit/byte işlemleri için CCS C derleyicisinde mevcut olan komutlar tanıtılmış ve ardından komutlara ilişkin örnek uygulamalar incelenmiştir.

6.1. PORT GİRİŞ/ÇIKIŞ KOMUTLARI

Port giriş/çıkış komutları port'ların giriş veya çıkış olarak kullanılıp kullanılmayacağı, port ve/veya port pin'lerinin çıkış durumlarını, port girişlerindeki bilgilerin okunması gibi işlemlerin gerçekleştirilmesini sağlayan komutlardır.

6.1.1. SET_TRIS_X() Komutu

Bu komut port pin'lerinin hangisinin giriş pin'i, hangisinin çıkış pin'i olacağını belirtir. Komuttaki **"X"** yerine işlem yapılacak port'un ismi (A, B, C, ...), parantez içine ise

heksadesimal olarak 8 bit'lik değer girilir. Pin değeri **"0"** girilmişse o pin'in çıkış pin'i olarak kullanılacağı, **"1"** girilmişse o pin'in giriş pin'i olarak kullanılacağı anlamındadır. Kullanımı;

set_tris_x(*değer*);

```
set_tris_b(0b00001111);
set_tris_b(0x0F);
```

şeklindedir. Bu iki komutta aynı anlamdadır. Sadece birinde değer binary olarak, diğerinde heksadesimal sayı olarak girilmiştir. Bu komut bize B port'unun B0, B1, B2, B3 pin'lerinin giriş olarak kullanılacağını, B4, B5, B6, B7 pin'lerinin ise çıkış olarak kullanılacağını bildirir.

6.1.2. GET_TRIS_X() Komutu

İstenen port'un TRIS kaydedicisi değeri ile geri dönen bir fonksiyondur. Komutta "X" yerine TRIS kaydedicisi değeri okunacak port'un ismi girilir.

```
set_tris_a(0xFE);
bilgi=get_tris_a();
```

yukarıdaki örnekte bilgi değişkeninin değeri "FE" olur.

6.1.3. OUTPUT_LOW() Komutu

Bu komut ile istenen port'un istenen bit'i lojik 0 yapılır. Yani istenen ucun çıkışı 0 (sıfır) yapılır. Komutta parantez içine işlem yapılacak pin ismi girilir.

output_low(*pin ismi*);

```
output_low(pin_A2);
```

komutu ile A port'unun A2 numaralı ucu lojik 0 yapılmış olur.

6.1.4. OUTPUT_HIGH() Komutu

Bu komut ile istenen port'un istenen bit'i lojik 1 yapılır. Yani istenen ucun çıkışı 1 (bir) yapılır. Komutta parantez içine işlem yapılacak pin ismi girilir.

output_high(*pin ismi);*

```
output_high(pin_B3);
```

komutu ile B port'unun B3 numaralı ucu lojik 1 yapılmış olur.

6.1.5. OUTPUT_BIT() Komutu

Bu komut ile istenen port'un istenen bit'i lojik 0 veya lojik 1 yapılır.

output_bit(*pin, değer);*

output_bit(PINC4, 1);

komutu ile C port'unun 4. bit'inin çıkışı lojik 1 yapılmıştır.

6.1.6. OUTPUT_X() Komutu

Bu komut ile bir port'un tüm bit'lerine tek komutla çıkış değerleri yüklenir.

output_*port ismi* (*değer);*

output_B(0x0F);

komutu ile A port'unun B0, B1, B2, B3 bit'lerinin çıkışları lojik1, B4, B5, B6, B7 bit'lerinin çıkışı ise lojik 0 yapılıyor (0x0F=00001111). Parantez içine yazılan değer 8 bit'lik bir sayı olmak zorundadır.

6.1.7. OUTPUT_TOGGLE() Komutu

Bu komut istenen port'un istenen bit'inin çıkışının durumunu değiştirir. Çıkış lojik-1 ise lojik-0, lojik-0 ise lojik-1 olur.

output_toggle(*pin ismi*);

output_toggle(pin_c4);

komutu ile C4 pin'inin çıkışı lojik-1 seviyesinden lojik-0 seviyesine getirilir.

6.1.8. OUTPUT_FLOAT() Komutu

Bu komut ile açık kolektör yapısına sahip (pull-up bağlantılar) bağlantılarda istenen pin giriş pin'i olarak ayarlanır. Böylece pull-up bağlantılı hat pull-up direnci ile high (lojik-1) seviyesinde tutulur.

output_float(*pin ismi*);

output_float(pin_b0);

6.1.9. INPUT() Komutu

Bu komut PIC mikrodenetleyicisinin istenen pin'inin durumunu okumaya yarar. Yani istenen giriş pin'inin lojik olarak değerini okur.

***değişken ismi* = input (*pin ismi*);**

bilgi=input(pin_A2);

komutu ile A2 pin'i girişinden dijital bilgi (0 veya 1) okunmakta ve bilgi değişkenine aktarılmaktadır.

6.1.10. INPUT_X() Komutu

Bu komut ile istenen giriş port'u durumu tümüyle okunur.

***değişken*=input_*port ismi* ( );**

data=input_c( );

komutu ile C port'unun komple durumu "data" adlı değişkene aktarılmış olunur. Değişken 8 bit'lik olmalıdır.

6.1.11. INPUT_STATE()

Bu komut ile istenen pin'in çıkış mı yoksa giriş mi olduğu öğrenilmektedir. Geri dönüş değeri "1" ise pin giriş, "0" ise pin çıkış olarak yönlendirilmiş demektir.

***değişken ismi* = input_state(*pin ismi*);**

i=input_state(pin_b2);

komutu ile B2 pin'inin giriş veya çıkış olarak mı yönlendirildiğini belirten bilgi i değişkenine aktarılmaktadır.

6.1.12. PORT_A_PULLUPS(), PORT_B_PULLUPS() Komutları

Bu komutlar ile A ve/veya B port'unun denetleyicide varsa pull-up fonksiyonunu aktif veya pasif hale getirmeye yarar.

Parantez içine aktiflik için **TRUE**, pasiflik için **FALSE** yazılır.

port_a_pullups(FALSE veya TRUE);, port_b_pullups(FALSE veya TRUE);

port_b_pullups(true); // B port'unun pull-up özelliği aktif hale getiriliyor.

6.2. BİT VE BYTE İŞLEMLERİ KOMUTLARI

Bit/Byte ile ilgili işlem yapan komutlar ile programımızda bit seviyesindeki işlemleri rahatlıkla yapabiliriz. Bu komutlarla istenen bir değerin (örneğin 0xA5 değeri) istenen bit'i üzerinde (örneğin değerin 5.bit'i) istenen değer verme ve kontrol döngüleri işlemleri gerçekleştirilebilir.

6.2.1. BIT_SET() Komutu

Bu komut istenen değişkenin istenen bit'ini "1 (bir)" yapar.

bit_set (*değişken, bit numarası*);

```
int x;
x=7;
bit_clear(x,3); // x'in yeni değeri şimdi 15 oldu.
```

Yukarıdaki örnekte x değişkeni tanımlanıyor ve içine 7 değeri atanıyor. Bit_set komutu ile de x değişkeninin 3.bit'i bir yapılıyor. Yani x değişkeni (7 = 0111) iken, 3.bit'i birleniyor ve (1111 = 15) değerini alıyor.

6.2.2. BIT_CLEAR() Komutu

Bu komut istenen değişkenin istenen bit'ini "0 (sıfır)" yapar.

bit_clear (*değişken ismi, bit numarası*);

```
int x;
x=6;
bit_clear(x,2); // x'in yeni değeri şimdi 2 oldu.
```

Yukarıdaki örnekte x değişkeni tanımlanıyor ve içine 6 değeri atanıyor. Bit_clear komutu ile de x değişkeninin 2.bit'i sıfır yapılıyor. Yani x değişkeni (6 = 0110) iken, 2.bit'i sıfırlanıyor ve (0010 = 2) değerini alıyor.

6.2.3. BIT_TEST() Komutu

Bit_test komutu ile istenen değişkenin istenen bit'inin değerinin durumu öğrenilir (Bit 0 mı?, 1 mi? o öğrenilir). Fonksiyon 0 veya 1 bilgisi ile geri döner.

bit_test (*değişken ismi, bit numarası*);

```
x=0x35;  // Heksadesimal olarak 35 değeri x değişkenine yükleniyor.
y=bit_test( x, 5 );
```

Yukarıdaki örnekte y değişkeni 1 olur. Çünkü x = 0x35 = 0011 0101 dir ve 5.bit'i 1'e eşittir.

6.2.4. SWAP() Komutu

Swap komutu 8 bit'lik bir sayısal değişkenin düşük dörtlü bit'leriyle (lower nibble), yüksek dörtlü bit'lerini (upper nibble) yer değiştirir.

swap (*değişken*);

```
int x;
x= 0x35;
swap (x); // x'in değeri şimdi 53 olur.
```

Yukarıdaki örnekte x = 35 = 0011 0101 dir. Swap(x) komutu ile düşük ve yüksek dörtlü bit gurubu yer değiştirir ve sonuçta x= 0101 0011= 53 olur.

6.2.5. MAKE8() Komutu

Make8 komutu, 16 veya 32 bit'lik sayıların istenen 8 bit'lik kısmını alır.

make8(*16 veya 32 bit'lik değişken, ofset değeri*);

Ofset değeri 16 veya 32 bit'lik değişkenin hangi 8 bit'lik kısmının alınacağını belirtir. Ofset değeri sadece 0, 1, 2, 3 olabilir.

```
int8   y;
int32  x;
x=0xA35D82EF;   // (32 bit'lik sayı. Her karakter 4 bit'liktir.)
y=make8( x, 2);   // y'nin değeri 5D olur.
```

Yukarıdaki örnekte x değişkenini 8 bit'lik guruplara bölersek x = A3 5D 82 EF olur.
Her iki karakter toplam 8 bit'tir. 3 2 1 0
Böylece örnekte kullanılan komut ile y=make8(x, 2);, x değişkeninin 8 bit'lik 2. grubunun değeri y değişkenine aktarılıyor. Bunun sonucunda y değişkeninin değeri 5D olur.

6.2.6. MAKE16() Komutu

Bu komut ile iki tane 8 bit'lik sayının birleşiminden 16 bit'lik sayı oluşturulur. Yani iki tane 8 bit'lik sayıyı birleştirir ve 16 bit'lik sayı elde eder.

***16 bit'lik değişken* = make16(*8 bit'lik değişken, 8 bit'lik değişken*)**

```
int8 a,b;
long x;
a=0x20;
b=0x12;
x=make16(a, b);  // x'in yeni değeri 0x2012 olur.
```

6.2.7. MAKE32() Komutu

Bu komut ile 32 bit'lik sayı oluşturulur. Komutta parantez içinde verilen değişken sayısı en fazla dört tane olabilir. Parantez içindeki değişkenler ister 8 bit'lik, ister 16 bit'lik, istenirse de 8 ve 16 bit'lik değişkenlerin birleşimi şeklinde olabilir.

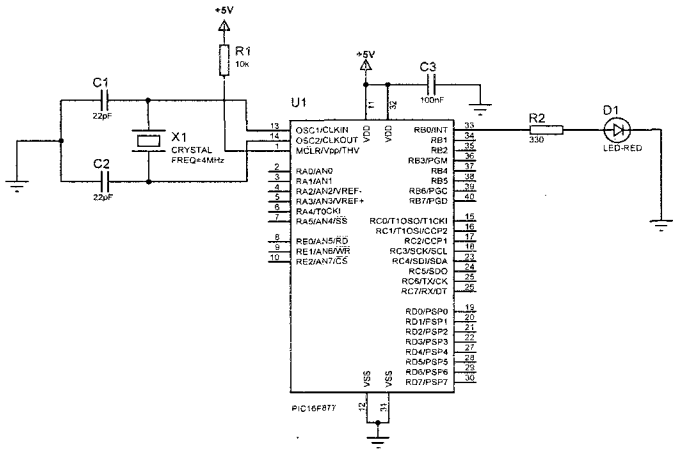
***32bit'lik değişken* = make32 (*değişen1, değişken2, değişken3, değişken4*);**

```
int8 y;
int16 z;
int32 x;
z=0x34de;
y=0xab;
x=make32(y,z,y);  // x'in değeri x = 0xab34deab olur.
x=make32(z,y);    // x'in değeri x = 0x0034deab olur.
```

6.3. LED YAKMA VE SÖNDÜRME UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu örnekte RB0 pin'ine bağlı bir LED'i belli aralıklarla yakıp söndüreceğiz. CCS C'deki ilk uygulamamız olduğu için böyle basit bir örnek ile başlanmıştır. Bu uygulama örneği aynı zamanda CCS C ile yazılan programlarda genel olarak neler yapıldığının da öğrenildiği bir çalışmadır. Uygulamanın devre şeması Şekil-6.1'de verilmiştir.



Şekil-6.1. *LED yakma ve söndürme uygulaması devre şeması*

Devre programı;

```
/*****************************************************
 Uygulama Adı        : PIC16F877 ile Led Yakma ve Söndürme Programı
 Programın Amacı     : RB0 pin'ine bağlı LED'i belli aralıklarla yakıp
                       söndürmek
 PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
*****************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         //dosyası tanıtılıyor.

// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //  osilatör frekansı belirtiliyor

#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli


/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı
```

```
   set_tris_b(0x00);        // B port'u komple çıkış olarak
                            // yönlendiriliyor.

   output_b(0x00);          // B port'u çıkışı komple sıfır yapılıyor.

   basla:                   // Etiket ismi.

   output_high(pin_b0);     // RB0 pin'i çıkışı lojik-1 yapılıyor.
   delay_ms(500);           // 500 ms gecikme veriliyor.
   output_low(pin_b0);      // RB0 pin'i çıkışı lojik-1 yapılıyor.
   delay_ms(500);           // 500 ms gecikme veriliyor.

   goto basla;              // basla etiketine dallanılıyor.
}
```

Programın yazımında ilk başta kullanılacak denetleyicinin başlık dosyası tanıtılmıştır. Bu komut her programda olmak zorundadır. Eğer uygulamanızda PIC16F877A denetleyicisini kullanıyorsanız başlık dosyasını 16F877A.h şeklinde değiştirmeniz gerekir. Başka denetleyici modelleri içinde aynı mantıkla başlık dosyası ismini değiştirmeyi unutmayınız.

```
#include <16f877.h>
```

Daha sonra ise kullanılan denetleyici ile ilgili konfigürasyon (yapılandırma) bit'leri tanımlamaları belirtilmiştir. Bu komutta her programda olmalıdır.

```
#fuses  HS,NOWDT,NOPROTECT,NOBROWNOUT,LVP,NOPUT,NOWRT,DEBUG,NOCPD
```

Bu programda bir sonraki bölümde ayrıntılı olarak göreceğiniz `#use delay(clock=4000000)` komutu kullanılmıştır. Bu komut, programda kullanacağımız gecikme komutlarında (delay_ms() vb.) derleyicinin kullanılan osilatör frekansını bilmesi için kullanılmaktadır. Bölüm 7'de bu komut ayrıntılı olarak anlatılmıştır.

PIC denetleyicilerde hangi port'un hangi pin'lerinin giriş veya çıkış olarak kullanılacağı programda bildirilmelidir. Bölüm 4.14'te anlatıldığı gibi CCS C derleyicisi bu işi kullanıcının isteğine göre otomatik yapmaya veya yapmamaya olanak sağlar. Bu örnekte `#use fast_io(b)` komutu kullanıldığından programda programcı kendisi port giriş/çıkış yönlendirmesini (set_tris_x() komutları ile) yapacaktır. CCS C de programda #use xxx_io() komutlarından biri kullanılmazsa, derleyici tarafından otomatik olarak #use standart_io() komutu kullanılmış sayılır. Bu komut sayesinde hangi port'un giriş hangi port'un çıkış olacağının programda bildirilmesine gerek yoktur. Çünkü derleyici otomatik olarak kullanılan komutlara göre port'ları giriş ve çıkış olarak yönlendirir. Fakat ilk örneklerde öğrenim açısından programlarda yönlendirme komutları kullanılacaktır.

Gerekli ön tanımlamalar bittikten sonra programda ana fonksiyon kısmına geçilmiştir. PIC16F877 denetleyicisi kendi içinde çeşitli donanım birimlerini (ADC, EEPROM, CCP vb.) barındırmaktadır. Sonuçta aynı pin'in birçok fonksiyonu

olabilmektedir. Program yazarken kullanılmayan pin özelliklerinin programda istenmeyen etkiler meydana getirmemesi için, **kullanılmayan özellikler program komutları ile pasif hale getirilmelidir. Bu nedenle ana fonksiyon başında kullanılmayan donanım birimleri pasif hale getirilmiştir.** Bu fonksiyonlar ilerideki uygulamalarda daha iyi anlaşılabilir. Eğer PIC16F877A denetleyicisi kullanıyorsanız veya karşılaştırıcı (comparator) modülü içeren başka bir denetleyici kullanıyorsanız karşılaştırıcı birimini kullanmadığınız uygulamalarda yukarıda anlatılan ana fonksiyonun (main ()) hemen altında yer alan komutlara `setup_comparator(NC_NC_NC_NC);` komutunu da ekleyiniz.

Programda hangi port ve pin'lerinin giriş veya çıkış olacağı da belirtilmiştir. Aşağıdaki komut ile B port'u komple çıkış olarak yönlendirilmiştir. Komuttaki "0x" ekinin heksadesimal gösterimler için kullanıldığını hatırlayalım (binary olarak göstermek isteseydik "0b" ön ekini kullanmak gerekliydi).

```
set_tris_b(0x00);
```

Programda ilk başta `output_b(0x00);` komutu ile B port'unun çıkışları komple sıfırlanmıştır. Böylece PIC'e ilk enerji geldiğinde yapılan sıfırlamada kaydedici içeriklerinin hepsinin sıfır olmama ihtimaline önlem alınmış olunmaktadır. Daha sonra `output_high(pin_b0);` komutu ile RB0 pin'i çıkışı lojik-1 yapılmıştır. Dolayısıyla LED yanmıştır.

NOT: Komutların ve komutlarda parantez içlerine yazılan CCS C derleyici tanımları yazımlarında büyük veya küçük harf kullanabilirsiniz. Örnek olarak parantez içine yazdığımız pin_b0 kelimesini PIB_B0 olarak da yazabilirdik. Fakat C dilinde değişkenlerin ve sabitlerim küçük büyük harf duyarlı olduğunu unutmayın.

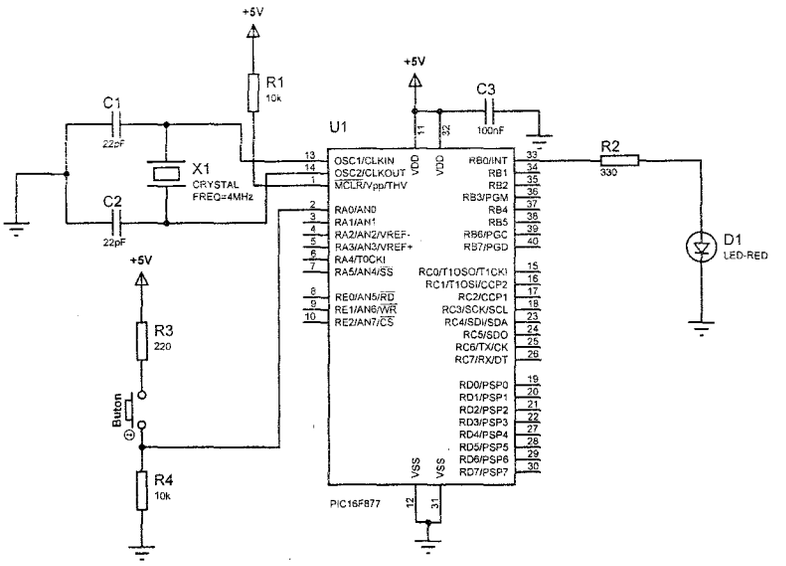
Ardından `delay_ms(500);` komutu ile de 500 ms'lik bir gecikme verilmiştir. Sonra `output_low(pin_b0);` komutu ile RB0 pin'i çıkışı lojik-0 yapılmıştır, yani LED sönmüştür. Ardından yine 500 msn'lik gecikme verilmiştir. Son olarak da `goto basla;` komutu ile de `basla:` etiketine dönülmüş (dallanılmış) ve bu sayede sonsuz bir döngü oluşturulmuştur. Normalde C dilinde dallanma komutları (goto) pek kullanılmaz. Bunların yerine kontrol döngü fonksiyonları kullanılır. İleriki bölümde kontrol döngüleri kısmı da incelendiğinde daha esnek programlar yapacağız.

Programı hem PROTEUS ISIS'de hemde deneme kartında deneyebilirsiniz. Deneme kartında programı denerken B port'u jumper'ını LED konumuna almayı unutmayın.

6.4. BUTON İLE LED KONTROL UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu örnekte PIC16F877 denetleyicisinin RB0 ucuna bağlı bir LED'i RA0 butonu ile yakıp söndüreceğiz. Butona basılınca LED yanacak, butondan elimizi çektiğimizde LED sönecektir. Devrenin açık şeması Şekil-6.2'de görülmektedir. Şekilden görüldüğü gibi butona basılmadığı zaman girişe lojik-0 bilgisi, butona basıldığında ise girişe lojik-1 bilgisi gelmektedir.



Şekil-6.2. *Buton ile LED kontrol uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Buton ile Led Kontrol Uygulaması
 Programın Amacı     : Bu program ile B0 port'una bağlı LED RA0
                       girişine bağlı buton ile kontrol edilmektedir.
 PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
 *****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.

// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use fast_io(a) // Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli
```

```
int1 x;     // 1 bit'lik x adında tamsayı tipinde bir değişken
            // tanımlanıyor.

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0b00000001);     // RA0 ucu giriş olarak ayarlandı.
   set_tris_b(0x00);     //RB0 port'u tümüyle çıkış olarak ayarlandı.

   basla:                    // basla etiketi tanımlanıyor.
   x=input(pin_a0);          // RA0 girişi okunuyor ve x
                             //değişkenine aktarılıyor.
   output_bit(pin_b0,x);     // x değişkeni değeri RB0 pin'i çıkışına
                             // aktarılıyor.
   goto basla;               // basla etiketinin olduğu yere gidilir.
}
//*****************************************************************
```

Programda önceki uygulamada olduğu gibi ilk başta kullanılan denetleyici başlık dosyası tanıtılmış, konfigürasyon bit tanımlamaları belirtilmiştir ve kullanılmayan donanım birimleri devre dışı bırakılmıştır. Bu kısımlar tüm programlarda aynı şekilde kullanılacaktır.

`#use fast_io(a)`, `#use fast_io(b)` komutlarıyla port giriş/çıkış yönlendirmelerinin otomatik olarak değil de programda komutlarla (set_tris_x() komutları ile) yapılacağı derleyiciye bildirilmiştir.

Bu örnekte bir de bir değişken tanımlanmıştır. `int1 x;` komutuyla 1 bit'lik (sadece 0 veya 1 değerini alabilir) tam sayı tipinde x isminde bir değişken tanımlanmıştır. Değişkenin tanımlandığı yere dikkat ediniz.

Programda hangi port ve pin'lerinin giriş veya çıkış olacağı da belirtilmiştir. Örnek olması amacıyla A ve B port'larının giriş ve çıkış olma komutları kullanılırken A port'u binary formda, B port'u ise genelde kullandığımız heksadesimal formda gösterilmiştir.

`set_tris_a(0b00000001);` , `set_tris_b(0x00);`

input(pin_a0); komutuyla RA0 pin'i girişi okunmuştur. Normalde butona basılmadığı zaman RA0 girişi lojik-0 seviyesindedir. Butona basıldığı zaman giriş lojik-1 olacaktır. `x=input(pin_a0);` komutu ile RA0 girişi okunmuş ve x değişkenine aktarılmıştır. Bu okunan değer ise `output_bit(pin_b0,x);` komutu ile RB0 çıkış

pin'ine aktarılmıştır. En son olarak ta `goto basla;` komutu ile de `basla` etiketine dönüş sağlanarak, programın sürekli dönmesi sağlanmıştır.

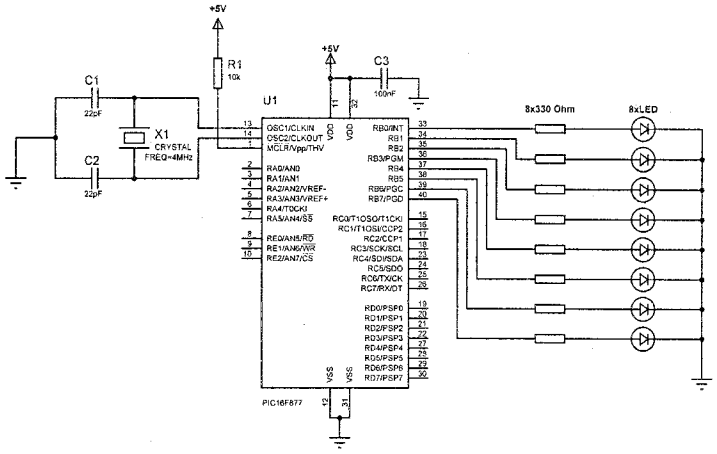
Bir butona basıldığında butondan gelen sinyalde bir ark oluşur. Bu arklar hızlı algılama yapabilen denetleyici tarafından butona birden fazla basıldı diye anlaşılabilir. Bu nedenle butona basılıp basılmadığını kontrol eden komuttan sonra yaklaşık 15-20ms (mili saniye) bir gecikme vermek gereklidir. Buton uygulamalarında buton girişi kontrol edilirken hemen arkasına buton arkını söndürmek için bir gecikme (genelde 15ms yeterlidir) veya bir kontrol döngüsü konulmalıdır. İleriki uygulamalarda buton arkı için koyulan gecikme ve kontrol döngüsü komutları örnekleri incelenmiştir.

Programı hem PROTEUS ISIS'de hemde deneme kartında deneyebilirsiniz. Deneme kartında programı denerken B port'u klemensini LED konumuna almayı unutmayın.

6.5. LED TOGGLE UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada output_toggle() komutu kullanılarak B port'una bağlı 8 adet LED'in durumu sırayla değiştirilmektedir. Devrenin bağlantı şeması Şekil-6.3'de görülmektedir. Devreye enerji verildiğinde ilk olarak RB0 ucuna bağlı LED yanacak ve belli süre aralıklarıyla sırayla diğer LED'ler yanacaktır. En son RB7 ucundaki LED yandıktan sonra bu sefer sırayla RB0'den itibaren tüm LED'ler sönmeye başlayacaktır. Tüm LED'ler söndükten sonra, RB0 ucundaki LED tekrar yanacak ve belli aralıklarla diğer LED'lerde tekrar yanmaya başlayacaktır. Programın çalışması bu şekilde devre enerjisi kesilene kadar devam edecektir.



Şekil-6.3. *LED Toggle Uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : PIC16F877 ile Led Toggle Uygulaması
 Programın Amacı     : Bu program ile B port'una bağlı 8 LED'i TOGGLE
                       komutu ile sırayla yakıp söndürmektedir.
 PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
 *****************************************************************/

#include <16f877.h> // Kullanılacak denetleyicinin başlık dosyası
                    // tanıtılıyor.

//Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //  osilatör frekansı belirtiliyor

#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli

/********* ANA PROGRAM FONKSİYONU********/

void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00);     //RB0 port'u tümüyle çıkış olarak ayarlandı.

   output_b(0x00);       // B port'u çıkışı sıfırlanıyor

   basla:

   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b0);    // RB0 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b1);    // RB1 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b2);    // RB2 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b3);    // RB3 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b4);    // RB4 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b5);    // RB5 ucunun çıkış durumu tersleniyor
   delay_ms(500);            // 500 ms'lik gecikme veriliyor
```

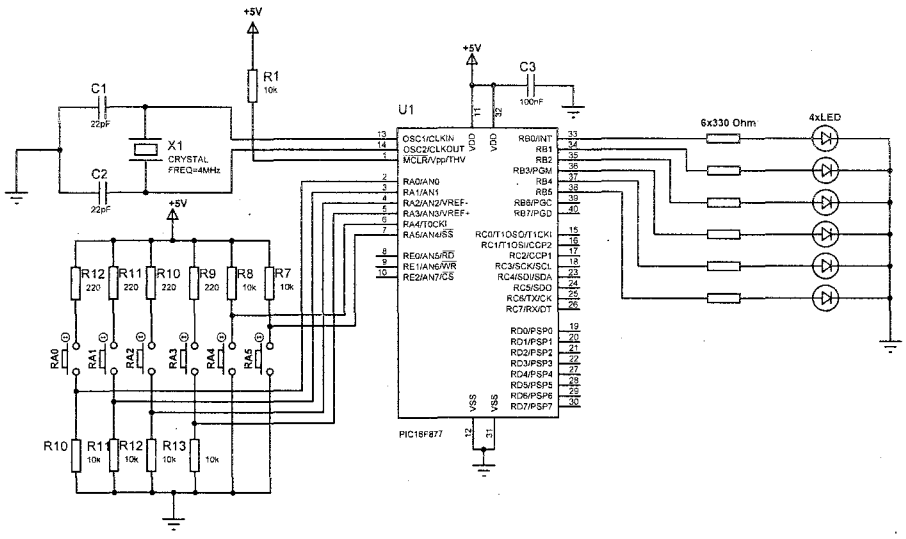
```
   output_toggle(pin_b6);  // RB6 ucunun çıkış durumu tersleniyor
   delay_ms(500);          // 500 ms'lik gecikme veriliyor
   output_toggle(pin_b7);  // RB7 ucunun çıkış durumu tersleniyor
   delay_ms(500);          // 500 ms'lik gecikme veriliyor

   goto basla;
}
//*****************************************************************
```

Programda önceki uygulamada olduğu gibi ilk başta yapılan tanımlamalar belirtilmiştir. Programda B port'u çıkış olarak ayarlandıktan sonra başlangıç olarak B port'u çıkışı `output_b(0x00);` komutu ile sıfırlanmıştır. Daha sonra sırayla `output_toggle(pin_b0);` komutlarıyla belirtilen pin'in çıkışı sahip olduğu konumun tersine döndürülmüştür. İlk başta RB0 pin'i lojik-0 olduğundan bu komut ile RB0 pin'i lojik-1 olacak ve LED yanacaktır. Daha sonra `delay_ms(500);`komutu ile 500 ms'lik gecikme verilmiştir. Tüm LED'lerin durumu sırayla bu şekilde değiştirilmektedir. RB7 ucundaki LED'de yanınca program `goto basla;` komutu ile `basla` etiketinin olduğu yere dallanmaktadır. Burada da tekrar toggle komutu ile RB0'ın çıkış durumu terslenmiştir. Yani en son RB0 çıkışı lojik-1'di, şimdi lojik-0 olacak yani sönecektir. Sönme işlemi 500 ms aralıklarla sırayla diğer LED'lerde de gözükecektir. Program bu şekilde sonsuz döngüde çalışmasına devam edecektir.

6.6. PORT OKUMA VE PORT'A BİLGİ GÖNDERME UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada A port'unun RA0, RA1, RA2 ,RA3, RA4 ve RA5 pin'lerine bağlı butonların durumları okunarak, okunan bu bilginin B port'una aktarılarak B port'una bağlı LED'lerde görünmesi sağlanmıştır. Devre şeklinde görüldüğü üzere RA0, RA1, RA2 ve RA3 butonlarına basılmadığında girişlere lojik-0, basıldığında ise girişlere lojik-1 bilgisi gelir. Çünkü bu butonlar lojik-1 de aktiftir. RA4 ve RA5 butonları ise lojik-0'da aktiftir. Yani bu butonlara basılmadığında girişlere lojik-1, basıldığında ise lojik-0 bilgisi gelir. Bu nedenle ilk anda hiçbir butona basılmadığında RB4 ve RB5 LED'leri yanık diğer tüm LED'ler sönük olacaktır. RA0, RA1, RA2 ve RA3 butonlarına basıldığında B port'undaki ilgili çıkış LED'i yanacak, RA4 ve RA5 butonlarına basıldığında ise ilgili çıkıştaki LED sönecektir.



Şekil-6.4. *Port okuma ve port'a bilgi gönderme uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Port Okuma ve Port'a Bilgi Gönderme
PIC PROG/DEKA        : Port B jumper'ı LED konumunda olmalı
 *****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin
                         //başlık dosyası tanıtılıyor.

// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor

#use fast_io(a) // Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli

#byte portb=6 //Bu komutla portb kelimesi denetleyicinin B port'u
                //kaydedicisi adresiyle eşleştirilmiştir

int giris;      // 8 bit'lik tamsayı tipinde değişken tanımlanıyor

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
```

```
   setup_adc_ports(NO_ANALOGS);     // ANALOG giriş yok
   setup_adc(ADC_OFF);              // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);             // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);             // CCP2 birimi devre dışı

   set_tris_a(0xFF); //A port'u tümüyle giriş olarak yönlendiriliyor
   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

   output_b(0x00);       // B port'u çıkışları sıfırlanıyor

   basla:
   giris=input_a();    // A port girişi komple okunuyor ve okunan
                       // değer giris değişkenine aktarılıyor
   giris=giris&0x0b00111111;//giris değerinin yüksek değerlikli son
                            //iki bit'i maskeleniyor
   portb=giris;        // giris değişkeni değeri B port'una çıkış
                       //olarak gönderiliyor
   goto basla;         // basla etiketine dallanılıyor
}
/*****************************************************************
```

Bu programda `#byte portb=6` komutu ile "portb" kelimesi PIC16F877 denetleyicisinin PORTB kaydedicisi adresine eşitlenmiştir. Yani derleyici artık portb değişkeni değerini 0x06 adresinde olan PIC denetleyicinin PORTB kaydedicisinin bulunduğu yere yazacaktır. Böylece portb değişkenine değer verilerek, B port'unun çıkışına direkt olarak bilgi gönderilebilir. Programda A port'undan okunacak bilginin kayıt edileceği değişken `int giris;` komutu ile tam sayı tipinde 8 bit'lik giris ismiyle tanımlanmıştır. `set_tris_a(0xFF);` komutu ile A port'u komple giriş olarak yönlendirilmiştir. B port'u da komple çıkış olarak yönlendirilmiştir.

Ana programda ilk başta `giris=input_a();` komutu ile A port'unun komple giriş değerleri okunmuş ve "giris" değişkenine aktarılmıştır. Daha sonra `giris=giris&0b00111111;` komutu ile "giris" değişkeni içeriğinin yüksek değerlikli son 2 bit'i maskelenmiş yani sıfırlanmıştır. Bu işlem "`giris`" değişkeninin 0b00111111 sayısı ile AND (&) işlemine tabi tutulmasıyla yapılmıştır. A port'u 6 bit'lik bir port'tur. B port'una gönderilen bilgi ise 8 bit'liktir. Dolayısıyla "giris" değişkeninin yüksek değerlikli son 2 bit'inin durumu A port'una bağlı değildir. Bu bit'ler maskeleme işlemi ile hep sıfırlanmıştır. Böylece B port'una bağlı LED'leri yakması engellenmiştir. Maskeleme işlemi birçok program algoritmalarında kullanılan pratik bir mantıktır. İleriki bölümlerde incelenen uygulamalarda OR işlemi ile yapılan maskeleme işlemleri de incelenmiştir.

Ardından `portb=giris;` komutu ile de `giris` değişkeni içeriği `portb` değişkenine atanmıştır. portb değişkeni de programda PIC'in B port'u adresine yerleştirildiğinden bu bilgi direkt olarak B port'u çıkışlarında görünecektir. Butonların bağlantısına dikkat edilirse butonlara basılmadığı zaman girişe lojik-0 bilgisi gelmekte, butona basılınca ise girişe lojik-1 bilgisi gelmektedir. Bu durumda basılan butonların karşılıkları B port'una bağlı LED'lerin yanmasıyla görülebilir. Aynı anda birden çok butona basıp, durumu LED'lerde gözleyebilirsiniz.

7. Bölüm

Karar Verme Döngü İşlemleri ve Gecikme Fonksiyonları

- IF Deyimi
- IF ELSE Deyimi
- SWITCH-CASE Deyimi
- FOR Döngüsü
- WHILE Döngüsü
- DO-WHILE Döngüsü
- CONTINUE Komutu
- GOTO Komutu
- DELAY_US()
- DELAY_MS()
- DELAY_CYCLE()
- LED Uygulamaları
- Buton Uygulamaları
- 7 Segment Display Uygulamaları

CCS C derleyicisi diğer C derleyicilerinde olan C komut kısıtlamalarına sahip değildir. Neredeyse tümüyle C dili ile sınırlama olmaksızın uyum içindedir. Bu bölümde de hem CCS C derleyicisinde hem de C dilinde kullanılan karar verme kontrol ve döngü komutları incelenmiştir. Döngü ve kontrol komutlarının anlatımının ardından çeşitli uygulamalarla bu komutların kullanıldığı örnekler verilmiştir. Ayrıca bu bölümde CCS C'de kullanılan gecikme (delay) fonksiyonları da incelenmiştir.

7.1. KARAR VERME DÖNGÜ DEYİMLERİ

Bu kısımda CCS C derleyicisinde kullanılan kontrol ve döngü fonksiyonlarının yapısı ve kullanımı üzerinde durulmuştur. Bahsedilen kontrol ve döngü fonksiyonlarının ANSI C'deki kullanımlarının ve özelliklerinin tümü CCS C içinde geçerlidir.

7.1.1. "if" Deyimi

if komutu bir veya birden fazla şartın gerçekleşmesine bağlı olarak bir komut veya komut bloğunun çalışmasını sağlar. if "eğer" anlamındadır. if komutunun basit söz dizimi aşağıdaki gibidir.

```
if  (şart)
      komut;
```

Burada belirtilen şart, bir değişken ifade veya sabit olabilir. Şart parantez içine yazılır. Eğer şarta bağlı olarak çalıştırılması istenen komutlar birden fazla ise, bu komutlar süslü parantez içine ({ }) içine alınır. if deyiminin ilk söz diziminde noktalı virgül kullanılmadığına dikkat edin. Aşağıda bu olay gösterilmiştir.

```
if  (şart)
{
komut ve/veya komutlar;
..........
}
```

if deyiminin icrasında, önce derleyici if parantezinin içindeki ifadenin sayısal değerini hesaplar. Hesapladığı bu sayısal değeri mantıksal Doğru ya da Yanlış olarak yorumlar. Yani 0 ise Yanlış, 0 dışında bir değer ise Doğru olarak yorumlar. Eğer ifadenin içindeki şart doğru ise şarttan sonra gelen komut veya komut bloğu çalıştırılır. Eğer ifadenin içindeki şart yanlış ise şart cümlesinden sonra gelen komut veya komut bloğu çalıştırılmaz, bir sonraki komuta geçilir.

7.1.2. "if else" Deyimi

"if else" deyimi "if" deyimiyle söz dizimi olarak aynıdır. Aşağıda söz dizimi belirtilmiştir.

```
if  (şart)
{
      komut veya komutlar ;
}
else
{
      komut veya komutlar ;
}
```

"if else" deyiminin icrasında, parantez içindeki şart doğru ise yani parantez içi mantıksal "1" ise parantezden sonraki komut veya komut bloğu çalışır, "else" kelimesinden sonraki komut veya komut bloğu çalışmaz. Eğer parantez içindeki şart yanlış ise yani parantez içi mantıksal "0" ise, parantezden sonraki komut veya komut bloğu çalışmaz, else kelimesinden sonraki komut veya komut bloğu çalışır.

7.1.3. "switch-case" Şartlı Dallanma Deyimi

Verilen ifade veya değişkenin aldığı değere göre programın akışını istenen seçeneklere yönlendirmek için kullanılır. Switch-case yapısının söz dizimi aşağıda verilmiştir.

```
switch (değişken veya ifade)
{
        case sabit1:
                        komut;
                          :
                        break;
        case sabit2:
                        komut;
                          :
                        break;
                          :
          case sabitN:
                        komut;
                        komut;
                          :
                        break;
        default:
                        komut;
                          :
                        break;
}
```

switch deyiminden sonra kullanılan ifadenin değeri, case ifadesinden sonraki değerlerden hangisine eşit ise oradaki komut veya komutlar işlem görür. Diğer case ifadeleri dikkate alınmazlar. Switch ifadesinden sonra verilen ifade hiçbir case sabitine eşit olmazsa program default seçeneğine gider ve oradaki komut veya komutları işler. Her case ifadesi işlendikten sonra switch komutundan çıkmak için break fonksiyonundan faydalanılır. **Switch kelimesinden sonra parantez içinde verilen değişken veya ifade char veya integer tipi bir değer olabilir. Parantez içindeki ifade string veya float tipi olamaz.** Aynı şekilde, case kelimesinden sonra da sadece integer veya char tipi sabit kullanılır. **Eğer case ifadesinden sonra char tipi sabit kullanılıyorsa ifade veya değişken tek tırnak içine alınır.**

7.1.4. "for" Döngüsü

for deyimi verilen şart doğru olduğu sürece belirtilen işlemlerin tekrarlanmasını sağlar. Aşağıda for döngüsünün söz dizimi görülmektedir.

```
for (başlangıç ; şart ; adım)
        {
                komut
                veya komutlar;
        }
```

Başlangıç: Döngüde kullanılacak değişkenlere ilk değerlerin atanması işlemleri burada yapılır. Döngünün ilk adımından önce değerlendirilir. Başlangıç ifadesinde aralarına virgül (,) koymak koşulu ile birden fazla ifade içerebilir.

Şart: Döngünün tekrarlanması için gerekli olan şart cümlesidir. Şart doğru olduğu sürece belirtilen komut veya komutlar tekrarlanır. Bir tek şart cümlesi olabileceği gibi, birden fazla şart cümlesi de yazılabilir. Bu durumda şart cümleleri arasına mantıksal operatörler kullanılmalıdır. Döngünün ilk adımında şart ifadesinin değeri yanlış ise döngü içerisindeki komut veya komutlar işlem görmez.

Adım: Döngünün her bir adımından sonra değerlendirilir. Döngünün artış / azalış miktarını belirtmek için kullanılır. Adım ifadesi aralılarına virgül (,) koymak sureti ile birden fazla ifade içerebilir.

Yukarıda belirtilen üç ifade de zorunlu değildir. Fakat aralarındaki ';' karakteri zorunludur. Aşağıdaki örneği inceleyelim.

```
for  ( ;  ; )
        printf ("Merhaba öğrenciler");
```

Yukarıdaki komut ile sonsuz sayıda bir döngü oluşturulmuştur. Bu nedenle ekrana sürekli "Merhaba öğrenciler" yazısını yazar.

for döngüsü içerisinde işlem görecek komut sayısı birden fazla ise komutlar blok içine { } parantezleri ile alınmalıdır. Bu parantezler arasına yazılan komutlar bir blok oluşturur ve döngünün her bir adımında bu blok içindeki tüm komutlar işlem görür. Tek komut için { } parantezlerini kullanmaya gerek yoktur.

7.1.5. "while" döngüsü

while döngüsü, verilen şart doğru olduğu sürece belirtilen işlemleri tekrarlamak için kullanılır. Şart kontrol işlemi döngüsünün başında yapılır. Aşağıda while döngüsünün söz dizimi verilmiştir.

```
while (şart)
      {
            komut veya komutlar;
      }
```

while deyiminde şart başlangıçta kontrol edilir. Eğer şart'ın sonucu doğru ise döngü içerisindeki işlemlere geçilir. Döngü içerisindeki işlemler yapıldıktan sonra, tekrar şart kontrolü yapılır ve şartın sonucunun doğru olması durumunda tekrarlama durumuna devam edilir. Bu işlemler şart doğru olduğu sürece tekrarlanır. Şartın sonucunun yanlış olması durumunda döngüden çıkılır. Başlangıçta şart yanlış ise döngü içerisindeki işlemler atlanır. Eğer şart doğru ise while ifadesinin yanındaki parantez içi 1 olur. Eğer yanlışsa parantez içi 0 olur. While (1) yazarsak her zaman için while döngüsü doğrudur anlamına geldiği için sonsuz döngü oluşturmuş oluruz.

Döngü içerisinde tek komut kullanılacaksa { } parantezlerini kullanmaya gerek yoktur.

7.1.6. "do-while" Döngüsü

Bu döngü komutu while komutu ile aynıdır. Fakat do-while döngüsünde şart döngünün sonunda test edilir. Bu durumda şartın sonucu başlangıçta yanlış olsa bile döngü bir kez işleme girecektir. Döngü içinde sadece bir komut yazılacaksa { } parantezlerini kullanmaya gerek yoktur. Aşağıda do-while döngüsünün söz dizimi verilmiştir.

```
do {
        komut veya komutlar;
    } while (şart) ;
```

7.1.7. "continue" Komutu

Döngü içerisinde döngünün bir sonraki adımına geçmesini sağlar. Kullanıldığı noktadan döngü sonuna kadar olan işlemleri atlar ve döngünün sonuna gider. Burada bir şart kontrolü varsa değerlendirilir, şart yoksa döngünün başına dönülür.

7.1.8. "goto" Komutu

Program akışını belirtilen program kesimine aktarmak için kullanılır. Ancak yapısal programlama dillerinde bu tür dallanmalar tercih edilmez. Aşağıda goto komutunun söz dizimi verilmiştir.

```
etiket;
        ......................;
        ......................;
                :
                :
        goto etiket;
```

7.2. GECİKME FONKSİYONLARI

Gecikme fonksiyonları istenen sürede gecikme sağlayan fonksiyonlardır. Gecikme fonksiyonları esnasında herhangi bir kesme meydana gelirse kesme de geçen süre gecikme fonksiyonu tarafından sayılmaz. Bu husus dikkate alınarak programda gecikme fonksiyonları kullanılmalıdır. Gecikme fonksiyonları kullanılmadan önce programın başında gecikme fonksiyonunun hesabında kullanılacak osilatör frekansının derleyiciye bildirilmesi gereklidir. Bu komut aşağıdaki şekildedir.

#use delay (clock= frekans cinsinden osilatör frekansı)

```
#use delay (clock=4000000)     // 4Mhz'lik osilatör için
#use delay (clock=20000000)    // 20 Mhz'lik osilatör için
```

7.2.1. DELAY_US() Fonksiyonu

Bu fonksiyon µsn (mikro saniye) biriminde gecikme sağlamaktadır. Fonksiyonda *zaman* kısmına 0-225 arası değerde bir değişken veya 0-65535 arasında bir tam sayı olarak istenen değer girilmelidir.

delay_us(*zaman*);

```
delay_us( 500 ); //  500 µsn'lik bir gecikme sağlar
```

7.2.2. DELAY_MS() Fonksiyonu

Bu gecikme fonksiyonu msn (mili saniye) biriminde gecikme sağlar. Fonksiyonda *zaman* kısmına 0-225 arası değerde bir değişken veya 0-65535 arasında bir tam sayı olarak istenen değer girilmelidir.

delay_ms(*zaman*);

```
delay_ms( 600 ); //  600 msn'lik bir gecikme sağlar.
```

7.2.3. DELAY_CYCLES() Fonksiyonu

Bu fonksiyon istenen komut sayısı süresi kadar gecikme sağlar. İşlemcide her komut osilatör frekansına bağlı olarak belli bir sürede işlenir. PIC denetleyiciler osilatör frekansını 4'e böler ve elde edilen frekans komutların işlenmesinde kullanılır. Fonksiyonda *komut sayısı* kısmına 1-255 arasında bir tam sayı girilir.

delay_cycles(*komut sayısı*);

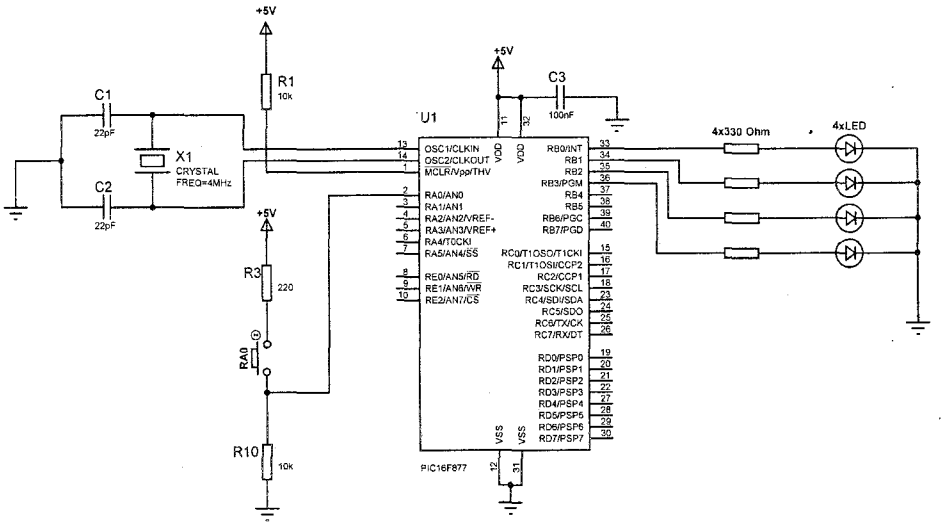
```
delay_cycles( 20 );     // 20 komutun işlenme süresi kadar gecikme sağlar.
```

7.3. BUTON İLE BCD SAYICI UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada B port'una bağlı (RB0, RB1, RB2, RB3) LED'ler ile BCD sayıcı uygulaması yapılmıştır. Bu örnek karar verme ve döngü işlemlerinin kullanıldığı ilk örneğiniz olacaktır. BCD sayılar 0'dan 9'a kadar olan desimal sayılar için 4 bit'lik

binary karşılık içerirler. Devre çalıştırıldığında her butona basılışta 0'dan 9'a kadar desimal sayılar artmakta ve BCD karşılıkları LED'lerde görünmektedir.



Şekil-7.1. *Buton ile BCD sayıcı uygulaması devre şeması*

Devre programı;

```
/*****************************************************************
Uygulama Adı        : Buton ile BCD Sayıcı
PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         //dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları******************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //  osilatör frekansı belirtiliyor

#use fast_io(a) // Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli

#define buton pin_A0      //"buton" adında "pin_A0" kelimesine eşit
                          // bir sabit tanımlanıyor
int i=1;      // 8 bit'lik tamsayı tipinde değişken tanımlanıyor

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
```

```
   setup_timer_1(T1_DISABLED);      // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);  // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);     // ANALOG giriş yok
   setup_adc(ADC_OFF);              // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);             // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);             // CCP2 birimi devre dışı

   set_tris_a(0x01);     // RA0 pin'i giriş olarak yönlendiriliyor
   set_tris_b(0x00);     //RB0 port'u çıkış olarak yönlendiriliyor

   output_b(0x00);       // B port'u çıkışları sıfırlanıyor

   while(1)        // Sonsuz döngü oluşturuluyor
   {
      if (input(buton))
      {
         delay_ms(15);  // Butona basılınca meydana gelen arkın
                        //etkisini önlemek için verilen gecikme
         output_b(i);   // b port'una i değeri yükleniyor
         i=i+1;         // i değeri 1 arttırılıyor
         while(input(buton));   // Buton bırakılana kadar bekle
         if (i==10)     // i değeri test ediliyor.
         {
            i=0;        // i değeri sıfırlanıyor
         }
      }
   }
}
//*********************************************************************
```

Devre programı incelendiğinde, her programda olduğu gibi ilk başta kullanılan denetleyici başlık dosyası ile konfigürasyon bit tanımlamaları tanıtılmış ve hemen ardından da programda kullanılan gecikme fonksiyonlarının doğru hesaplama yapabilmesi için devrede kullanılan osilatör frekansı belirtilmiştir. Genel tanımlama komutlarından sonra `#define buton pin_A0` komutu ile "buton" adında "pin_A0" değerinde bir sabit tanımlanıyor. Yani bu komuttan sonra derleyici artık "buton" kelimesini gördüğü yere "pin_A0" yazacaktır. Daha sonra `int i=1;` komutu ile "i" adında tamsayı tipinde 8 bit'lik bir değişken tanımlanmış ve ilk değer olarak 1 değeri yüklenmiştir.

Ana fonksiyonda (main ()) ilk başta kullanılmayan donanım birimleri pasif hale getirilmiştir. Ardından programda `#use fast_io(b)`, `#use fast_io(b)` komutları kullanıldığından `set_tris_a(0x01);`, `set_tris_b(0x00);` komutları ile port giriş/çıkış yönlendirilmeleri yapılmıştır. Eğer `#use fast_io(x)` komutu programda kullanılmasaydı `set_tris_x()` komutu ile port giriş/çıkış'larının bildirilmesine gerek olmazdı. Çünkü o zaman bu işi derleyici otomatik olarak yapacaktı.

`output_b(0x00);` komutu ile de başlangıçta B port'u çıkışı sıfırlanmıştır. `while(1)` ile sonsuz bir döngü oluşturulmuştur. While döngüsünün parantez içine yazılan şartı komutta hep "1" olduğundan döngü hep doğru olacak ve sonsuza kadar aynı

döngü dönecektir. Bu while döngüsü { parantezi ile başlayan ve aynı hizadaki } parantezi ile biten kısımdaki tüm komutların oluşturduğu bir döngüdür. Sonsuz döngülerde parantez içine "1" yerine "true" ifadesi de yazılabilirdi. İkisi de döngü şartının doğru olduğu anlamına gelir.

`if (input(buton))` komutu ile buton girişi yani RA0 girişi okunmakta ve butona basıldığında yani RA0 girişi lojik-1 olduğunda if döngüsünün aktif olması sağlanmıştır. Butona (RA0'a bağlı) basılınca giriş 1 olacak ve `input(buton)` komutu 1 değeri ile geri dönecektir. Dolayısıyla if komutunun içi **if (1)** olacak ve bu if döngüsüne ait komutlar işlenecektir. Eğer butona basılmamışsa bu if döngüsü komutları komple atlanacaktır. Böylece program butona basılmadığı sürece boşta dönmeye devam edecektir.

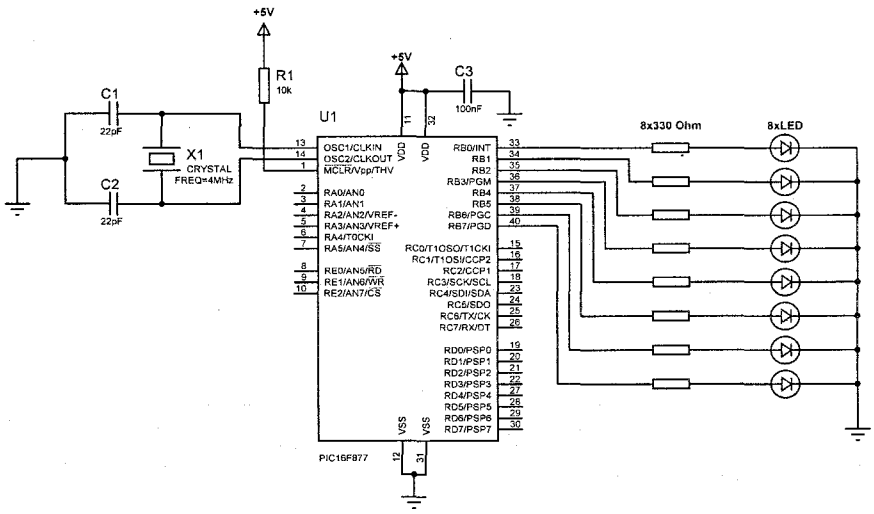
Butona basıldığını farz edersek if döngüsü komutları icra görmeye başlayacaktır. İlk komut olan `delay_ms(15);` komutu ile buton arklarını önlemek için gerekli bir 15 mili saniyelik gecikme verilmiştir. Ardından `output_b(i);` komutu ile de i değişkeni değeri b port'una gönderilmiştir. i değişkeni ilk değeri a olduğundan, bu komut ile RB0'a bağlı LED yanacaktır. `i=i+1;` komutu ile de i değişkeni değeri bir arttırılmıştır. Bu komut yerine i++; komutu da kullanılabilirdi. `while(input(buton));` komutuyla da buton'dan elimizi çekene kadar programın bu döngüden çıkmayarak başka bir işlem yapması önlenmiştir. Dolayısıyla bu komut sayesinde butona bir kez basılıyor ve BCD sayılarda sadece bir artış oluyor. Butona basmaya devam etsek dahi ikinci bir artış olmayacaktır. Bu döngüden elimizi butondan çekinceye kadar çıkılmayacaktır. Çünkü butona basılınca giriş değeri 1 olacak ve komut **while (1)** halini alacaktır. Yani döngü şartı doğru olmuş olacağından sürekli dönecektir. Ne zaman butondan elimizi çektiğimizde giriş 0 olacak ve komut **while (0)** halini alacaktır. Yani döngü şartı yanlış olduğundan döngüden çıkılacaktır. Aşağıdaki if döngüsü ile de, i değişkeni kontrol edilmekte ve i değişkeni 10'a eşitlenmektedir. Program bu şekilde çalışmasına devam etmektedir.

```
if (i==10)
        {
            i=0;
        }
```

7.4. YÜRÜYEN IŞIK UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Yürüyen ışık devresi diğer bir ismi ile kara şimşek devresi denetleyici çıkışına bağlı LED'lerin sırayla yanıp sönmesi ve tekrar bu döngünün başa dönmesi prensibine dayanır. Bu örnek için PIC16F877 denetleyicisinin B port'una 8 adet LED bağlanmıştır.



Şekil-7.2. *Yürüyen ışık uygulaması devre şeması.*

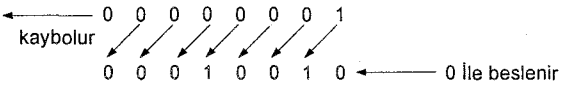
Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Yürüyen Işık
 PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h> // Kullanılacak denetleyicinin başlık
                    //dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları*******************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor

#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli

int i, led;          // i ve led isminde tam sayı tipinde 8 bit'lik
                     // değişkenler tanıtılıyor.
/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor
```

```
    output_b(0x00);        // B port'u çıkışları sıfırlanıyor

    while(1)
    {
       led=1;        //led değişkeni değeri 1 oluyor

       for(i=0;i<=7;i++)    // for döngüsü oluşturuluyor
       {
          output_b(led);  // led değişkeni değeri B port'una gönder
          led=led<<1;     // led değişkeni değerini 1 bit sola kaydır
          delay_ms(100);  // 100 msn'lik gecikme veriliyor
       }

       led=0x80;        //led değişkenine değer yükleniyor
       output_b(led);   //B port'una 0b10000000 bilgisi gönderiliyor
       delay_ms(100);   // 100 msn'lik gecikme veriliyor

       for(i=0;i<=6;i++) // for döngüsü oluşturuluyor
       {
          led=led>>1;     // led değişkeni değerini 1 bit sağa kaydır
          output_b(led);  // led değişkeni değeri B port'una gönder
          delay_ms(100);  // 100 msn'lik gecikme veriliyor
       }
    }
}
//*****************************************************************
```

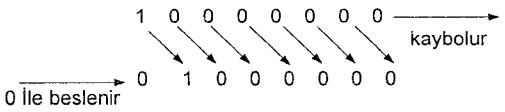
Programda ilk başta diğer uygulamalarda olduğu gibi gerekli denetleyici başlık dosyası ve konfigürasyon bit'leri tanıtılmıştır. Programda `int i,led;` komutuyla i ve led adında 8 bit'lik tam sayı tipinde değişkenleri tanımlanmıştır. Programda daha sonra `Set_tris_b(0x00);` komutu ile port giriş/çıkış yönlendirilmesi yapılmıştır. Bu komutla B port'unun komple çıkış olacağı belirlenmiştir.

`output_b(0x00);`komutu ile de başlangıçta B port'u çıkışı sıfırlanmıştır. `while(1)` ile sonsuz bir döngü oluşturulmuştur. Başlangıçta led değişkeni değeri `led=1;` komutu ile 1'e eşitlenmiştir. Ardından `for(i=0;i<=7;i++)` komutu ile bir **for döngüsü** oluşturulmuştur. Bu komutla; i değişkeninin sıfırdan başlayacağı, döngünün i değişkeni değerinin 7'den küçük veya eşit olduğu sürece devam edeceği ve her döngü başına gelindiğinde i değerinin 1 arttırılacağı istenmiştir. Döngüde `output_b(led);` komutu ile led=1 değeri B port'una gönderilmiştir. 1 değerinin 8 bit'lik binary açılımı 0b00000001'dir. Dolayısıyla RB0 ucuna bağlı LED bu komutla yanacaktır. Bu komuttan sonra `led=led<<1;` komutuyla led değişkeni değeri bir bit sola kaydırılmıştır. Şekil-7.3'de sola kaydırma olayı daha net anlaşılabilir.



Şekil-7.3. *Sola kaydırma işlemi.*

LED'lerin yanıp söndüğünü rahatça görebilmek için `delay_ms(100);` komutuyla 100 mili saniyelik gecikme verilmiştir. Döngü i değerini 1 arttıracak ve başa dönecektir. Döngü i değeri 7'ye eşit veya küçük olduğu sürece devam edecektir. Döngüdeki komutlar tekrar işlem görecektir. i değişkeni değeri bir bit sola kaydırılınca 00000010 değerini almıştı. Yani bu defa RB1 ucuna bağlı LED yanacak diğerleri sönecektir. i değeri 8'e eşit olunca for döngüsünden çıkılacaktır. LED'lerin yanmasına RB7 ucuna bağlı son LED'den başlanması için `led=0x80;` komutu ile led değişkenine 0x80 yani binary olarak 0b10000000 değeri yüklenmiştir. Ardından `output_b(led);` komutu ile de bu değer B port'una gönderilmiş ve RB7 ucuna bağlı LED yakılmıştır. İkinci for döngüsünde ise led değişkeni değeri bu sefer sağa kaydırılarak LED'lerin yanma sırası tersine işletilmiştir. Diğer komutlar ilk for döngüsündeki ile aynıdır. Program bu şekilde sonsuza kadar devam edecektir. Programdaki gecikme süresi azaltılıp arttırılarak LED'lerin sırayla yanma süresi isteğe göre ayarlanabilir.

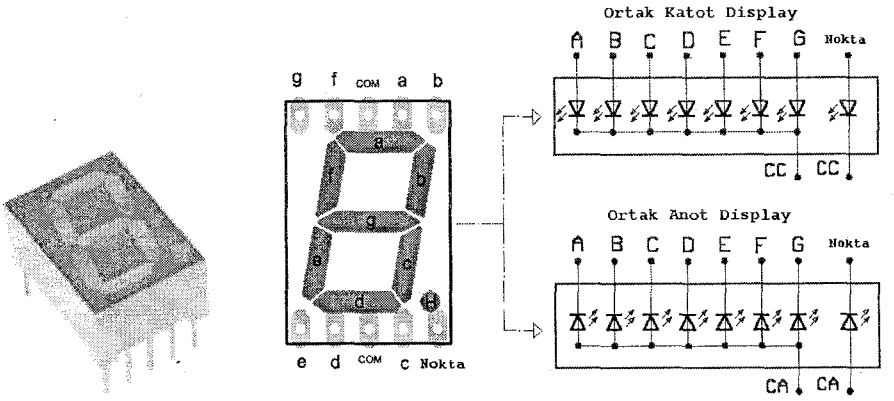


Şekil-7.4. *Sağa kaydırma işlemi.*

7.5. DISPLAY UYGULAMASI-1(0-F SAYICI)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada 7 segment bir display'in denetleyici ile kontrolü incelenmiştir. Bunun için 7 segment display'de 0 dan F'ye kadar sayan bir sayıcı yapılmıştır. Şekil-7.5'de 7 segment bir display, Şekil-7.6'da ise display segment harfleri, uçları ve iç bağlantıları görülmektedir. Display'ler ortak anot (Common Anode - CA) veya ortak katot (Common Cathode - CC) olmak üzere iki çeşit üretilmektedir. Şekil-7.6'daki uç bağlantıları iki çeşit display için de aynıdır. Sadece ortak uç olan COM ucu katot display'lerde (-), anot display'lerde (+) beslemeye bağlanmalıdır. Şekilden görüldüğü gibi her segment aslında bir LED yapısına sahiptir. Örneğin ortak katot display'de istenen bir segment'in yanmasını sağlamak için ortak uç şaseye bağlanır, daha sonra istenen segment ucuna (a, b, c, d, e, f, g, nokta segment'i h) pozitif enerji verilir (ortak anot için tam tersi). Böylece istenen segment yanar.



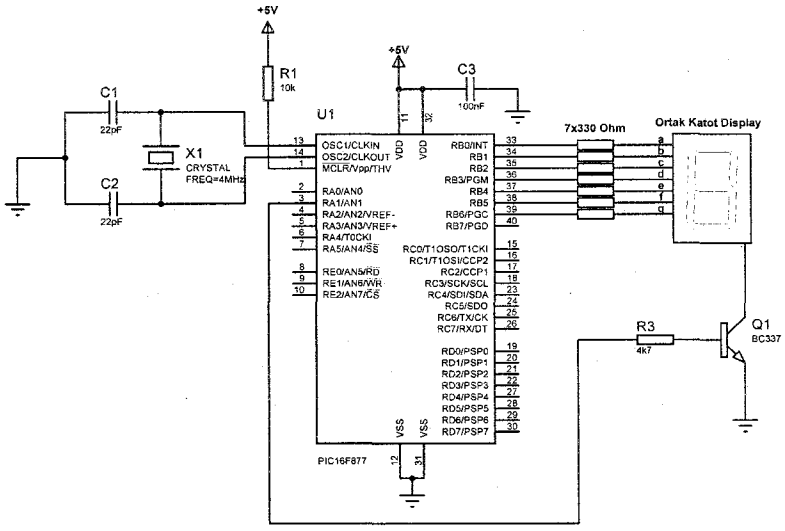
Şekil-7.5. *7 segment display.* ***Şekil-7.6.*** *Display segment isimleri ve bağlantıları.*

Display'de istenen bir sayının görülmesi için hangi segment'ler yanması gerekiyorsa o segment'lere aynı anda enerji verilmelidir. Örneğin display'de 5 rakamını göstermek için, display'in a, f, g, c, d segment'lerine lojik-1 (+5V) verilmesi gerekir. Tablo-7.1.'de ortak katot display'lerde hangi karakter için hangi bilginin display'e gönderilmesi gerektiği görülmektedir. Tablo'da display'in nokta (h) ucu kullanılmamıştır. Eğer display'in noktasının da yanması isteniyorsa h segment'ine de ortak katot display için 1 bilgisi, ortak anot display için 0 bilgisi uygulanmalıdır. Tabi bu durumda tabloda ki heksadesimal display bilgilerinin değişeceğini unutmamalısınız.

Sayı	Ortak Katot Display									Ortak Anot Display								
	h	g	f	e	d	c	b	a	HEX	h	g	f	e	d	c	b	a	HEX
0	x	0	1	1	1	1	1	1	0x3F	x	1	0	0	0	0	0	0	0x40
1	x	0	0	0	0	1	1	0	0x06	x	1	1	1	1	0	0	1	0x79
2	x	1	0	1	1	0	1	1	0x5B	x	0	1	0	0	1	0	0	0x24
3	x	1	0	0	1	1	1	1	0x4F	x	0	1	1	0	0	0	0	0x30
4	x	1	1	0	0	1	1	0	0x66	x	0	0	1	1	0	0	1	0x19
5	x	1	1	0	1	1	0	1	0x6D	x	0	0	1	0	0	1	0	0x12
6	x	1	1	1	1	1	0	0	0x7C	x	0	0	0	0	0	1	1	0x02
7	x	0	0	0	0	1	1	1	0x07	x	1	1	1	1	0	0	0	0x78
8	x	1	1	1	1	1	1	1	0x7F	x	0	0	0	0	0	0	0	0x00
9	x	1	1	0	1	1	1	1	0x6F	x	0	0	1	0	0	0	0	0x10
A	x	1	1	1	0	1	1	1	0x77	x	0	0	0	1	0	0	0	0x08
B	x	1	1	1	1	1	0	0	0x7C	x	0	0	0	0	0	1	1	0x03
C	x	0	1	1	1	0	0	1	0x39	x	1	0	0	0	1	1	0	0x46
D	x	1	0	1	1	1	1	0	0x5E	x	0	1	0	0	0	0	1	0x21
E	x	1	1	1	1	0	0	1	0x79	x	0	0	0	0	1	1	0	0x06
F	x	1	1	1	0	0	0	1	0x71	x	0	0	0	1	1	1	0	0x0E

Tablo-7.1. *Ortak katot display için değerler.*

Eğer örneklerinizde ortak anotlu bir dispay kullanacaksanız, Tablo-7.1'de verilen ortak anot kısmındaki verileri kullanmanız gerekir. Bu örnekte ortak katotlu 7 segment display kullanılmıştır ve B port'una bağlanmıştır. Devrenin açık şeması Şekil-7.7'de verilmiştir.



Şekil-7.7. *Display uygulaması-1 devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Display Uygulaması-1 (0-F Sayıcı)
 PIC PROG/DEKA       : Port B jumper'ı 7-seg konumunda olmalı
*****************************************************************/
#include <16f877.h> // Kullanılacak denetleyicinin başlık
                    //dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları************
#fuses
XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NONOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor
#use fast_io(a) // Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli

int i;     // i isminde 8 bit'lik tamsayı tipinde bir değişken
          //tanıtılıyor.

// Ortak katot display için veri değerleri
const int digit[16]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07,
0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
```

```
/********* ANA PROGRAM FONKSİYONU********/

void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x00); //A port'u tümüyle çıkış olarak yönlendiriliyor
   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

   output_b(0x00); // B port'u çıkışları sıfırlanıyor
   output_a(0x02); // Display'i aktif etmek için RA1 ucu lojik-1,
                   // diğer A port'u pin çıkışları lojik-0 yapılıyor

   while(1)    // Sonsuz döngü oluşturuluyor
   {
      for(i=0;i<=15;i++)
      {
         output_b(digit[i]); // digit[i] değerini B port'una gönder
         delay_ms(500);      // 500 msn bekle
      }
   }
}
//*****************************************************************
```

Programda sayaç olarak kullanılmak üzere i değişkeni tanıtılıyor. Ardından `const int digit[16]` komutuyla, B port'una display'de görünmesi için gönderilecek verilerin tutulduğu "digit" isminde tam sayı tipinde bir dizi tanımlanmıştır. Dizi üyeleri değişken değil sabit olduğundan tanımlanan dizi `const` komutu ile sabit şeklinde tanımlanmıştır. Programdaki `output_a(0x01);` komutu ile RA0 çıkışı lojik-1 yapılış, dolayısıyla Q1 transistörü iletime geçirilmiştir. Bu sayede display'in ortak katot ucu şase'ye bağlanmış olacaktır. Tek bir display'i sürerken ortak katotu transitör ile sürmek yerine direkt olarak şaseye de bağlayabilirdiniz. PIC/PROG deneme kartının bağlantısı bu şekilde olduğundan devrede transistör kullanılmıştır.

`for(i=0;i<=15;i++)` komutu ile i değişkeninin 0'dan başlayıp her döngü dönüşü 1 arttırılarak 15'den küçük veya eşit olduğu sürece dönmesi istenmiştir. Böylece display'de 0'dan F'e (15) kadar olan sayılar görüntülenecek ve tekrar başa dönülecektir. Döngü içinde `output_b(digit[i]);` komutu ile digit dizisinin i. elemanı B port'una gönderilmiş ve `delay_ms(500);` komutu ile de 500 msn'lik gecikme verilmiştir. Örneğin ilk başta i değeri sıfır olacağından digit[0]=0x3F değerini alacak ve bu değer B port'una gönderildiğinde display'de "0" değeri görünecektir. Döngünün ikinci dönüşünde i değeri bir arttırılacak ve değeri 1

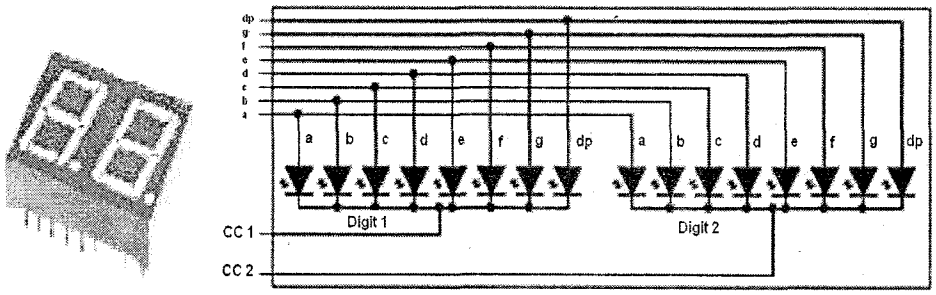
olacağından digit dizisinin birinci elemanı digit[1]=0x06 değerini alacaktır. Bu değerde B port'una gönderildiğinde display'de "1" değeri görünecektir. Bu işlemler bu şekilde devam edecektir.

Uygulama devrenizi PIC PROG/DEKA PIC16F87x deneme kartınızda denerken Port B Jumper'ını 7 seg konumuna almayı unutmayınız.

7.6. DİSPLAY UYGULAMASI-2(BUTON İLE 0-99 YUKARI SAYICI)

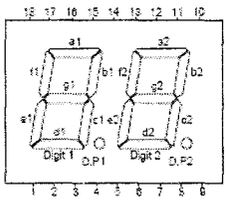
Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu örnekte ikili tümleşik display kullanılarak 0'dan 99' kadar sayan bir sayıcı yapılmıştır. Mantık olarak çoklu display'leri sürmek için ilk display uygulamamızda ki mantığı kullanırsak her display'e 7 adet bilgi ucunun gerekli olduğu görülür. Örneğin 4 dijitli bir display'de toplam 28 adet bilgi gönderme ucuna ihtiyaç duyulur ki bu çoğu gelişmiş denetleyicilerin neredeyse tüm pin'lerini işgal eder. Bu nedenle birden fazla display'i sürmek için, display'ler aynı bilgi hatları ile beslenir fakat display'ler sıra ile aktif edilir. Örneğin display'ler de "25" sayısını göstermek için; ilk başta ilk display aktif edilir ve bilgi hattına "2" sayısı gösterilir daha sonra ilk display pasif hale getirilir ikinci display aktif edilir ve bilgi hattına "5" sayısı gösterilir. Bu şekilde elde edilen görüntü tam net olmayabilir. Bunun nedeni display'deki segment'lerin fosfor tabakaları tam olarak aydınlanmadan diğer display'in aktif olmasıdır. Bu problemi ortadan kaldırmak için **tarama işlemi** yapılır. Tarama işleminde her display kısa bir süre aynı bilgi ile beslenir ve daha sonra diğer display'e geçilir. Bu işlem saniyede 16'dan fazla yapılırsa insan gözü bu yanıp sönmeleri durağan zanneder. Bu uygulamada ikili tümleşik ortak katot bir display'de tarama işlemini göreceğiz. Şekil-7.8'de ikili tümleşik bir display ve ortak katot ikili tümleşik bir display'in iç bağlantısı görülmektedir.



Şekil-7.8. *İkili tümleşik display ve ikili ortak katotlu display iç bağlantısı.*

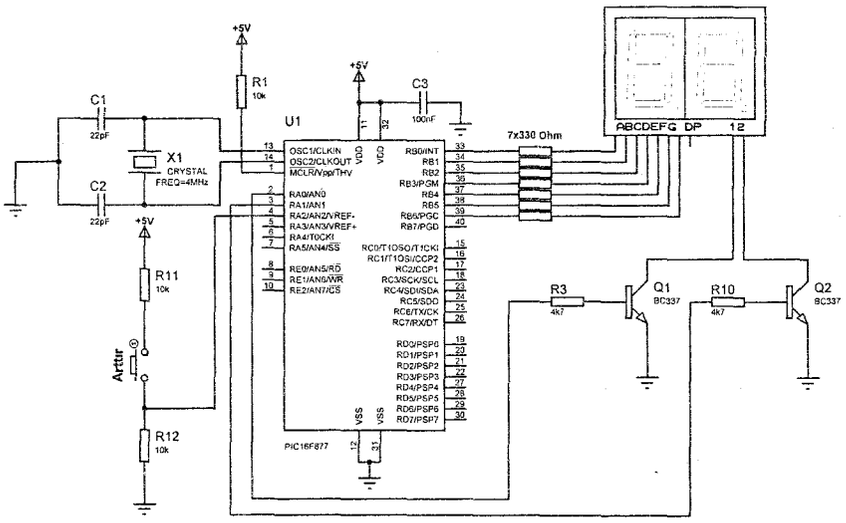
Şekil-7.9'da ikili tümleşik display'in pin numaraları ve segment isimleri, Tablo-7.2'de ise pin'lerin fonksiyonları görülmektedir. Şekil-7.9'da verilen çizim ortak katot ve ortak anot display'ler için aynıdır. Sadece ortak uca (COM-Common) ortak katot display için (-), ortak anot display için (+) gerilim verilmelidir. Uygulama devresi Şekil-7.10'da görülmektedir.



Şekil-7.9. *İkili display pin ve segment'leri*

Pin No	Pin Fonksiyonu	Pin No	Pin Fonksiyonu
1	e1 segment'i	10	b2 segment'i
2	d1 segment'i	11	a2 segment'i
3	c1 segment'i	12	f2 segment'i
4	1. Digit nokta ucu DP1	13	2. Digit ortak ucu
5	e2 segment'i	14	1. Digit ortak ucu
6	d2 segment'i	15	b1 segment'i
7	g2 segment'i	16	a1 segment'i
8	c2 segment'i	17	g1 segment'i
9	2. Digit nokta ucu DP2	18	f1 segment'i

Tablo-7.2. *İkili display pin fonksiyonları.*



Şekil-7.10. *Display uygulaması-2 devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Display Uygulaması-2 (Buton İle 0-99 Sayıcı)
 PIC PROG/DEKA       : Port B jumper'ı 7-seg konumunda olmalı
*****************************************************************/
```

```
#include <16f877.h> // Kullanılacak denetleyicinin başlık
                    // dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              // osilatör frekansı belirtiliyor

#use fast_io(a) // Port yönlendirme komutları a port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları b port'u için geçerli

#define display_1 pin_a0//display_1 ifadesi pin_a0 ifadesine eşleniyor
#define display_2 pin_a1//display_2 ifadesi pin_a1 ifadesine eşleniyor
#define buton     pin_a2  // buton ifadesi pin_a2 ifadesine eşitleniyor

// char tipinde değişkenler tanımlanıyor
char birler=0, onlar=0, sayi=0;

// Ortak katot display için veri değerleri
const int digit[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07,
0x7F, 0x6F};

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0xFC); // RA0,RA1 çıkış,diğer A port'u pin'leri giriş
   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

   output_low(display_1);       // 1. display pasif durumda
   output_low(display_2);       // 2. display pasif durumda

   while(1)  // Sonsuz döngü oluşturuluyor
   {
         birler=sayi%10;          // birler hanesi hesaplanıyor
         onlar=sayi/10;           // onlar hanesi hesaplanıyor

         output_high(display_2);  // 2. display aktif durumda
         output_b(digit[birler]); // display'e bilgi gönderiliyor
         delay_ms(5);             // gecikme veriliyor
         output_low(display_2);   // 2. display pasif durumda
         output_high(display_1);  // 1. display aktif durumda
         output_b(digit[onlar]);  // display'e bilgi gönderiliyor
```

```
        delay_ms(5);                // gecikme veriliyor
        output_low(display_1);      // 1. display pasif durumda

        if (input(buton))   // Arttır butonuna basıldı ise
        {
           delay_ms(10);    // Buton arklarını önleme gecikmesi

           sayi++;          // sayi değişkenini 1 arttır

           if(sayi==100)// Eğer sayi değişkeni 100 oldu ise sıfırla
             sayi=0;

           birler=sayi%10;          // birler hanesi hesaplanıyor
           onlar=sayi/10;           // onlar hanesi hesaplanıyor

           while(input(buton))//Buton bırakılana kadar döngüyü işle
           {
              output_high(display_2);  // 2. display aktif durumda
              output_b(digit[birler]);//displaye bilgi gönderiliyor
              delay_ms(5);             // gecikme veriliyor
              output_low(display_2);   // 2. display pasif durumda
              output_high(display_1);  // 1. display aktif durumda
              output_b(digit[onlar]); //displaye bilgi gönderiliyor
              delay_ms(5);             // gecikme veriliyor
              output_low(display_1);   // 1. display pasif durumda
           }
        }
    }
}
//*****************************************************************
```

Programda denetleyici başlık dosyası, konfigürasyon ayarları, gecikme fonksiyonu için kullanılan osilatör frekansını belirleme gibi genel tanımlamalardan sonra `#use fast_io(a)`, `#use fast_io(b)` komutları ile de port yönlendirmelerin otomatik olarak değil de, program komutları ile yapılacağı belirtilmiştir.

Devre şemasında görüldüğü üzere ikili display'in iki tane seçme ucu vardır. Birinci display'i seçen uç RA0 ucu, ikinci display'i seçen uçta RA1 ucudur. Program yazarken akılda kalması ve kolaylık sağlaması nedeniyle bu pin'lerin yerine geçecek sabitler tanımlanmıştır. `#define display_1 pin_a0` , `#define display_2 pin_a1`, `#define buton pin_a2` komutları ile PIC denetleyicinin RA0 pin'i "display_1" kelimesine, RA1 pin'i "display_2", RA2 pin'i ise "buton" kelimesine eşitlenmiştir. Yani sabit tanımlaması yapılmıştır. Bu komutlardan sonra artık derleyici "display_1" kelimesini pin_a0, "display_2" kelimesini pin_a1, "buton" kelimesini pin_a2 ifadesi olarak algılayacaktır.

`char birler=0, onlar=0, sayi=0;` komutlarıyla değişkenler tanımlanmış ve ilk değer olarak sıfır yüklenmiştir. Karakter tipinde tanımlanan birler ve onlar değişkenleri sayılacak sayının birler ve onlar basamağının değerlerini tutmak içindir. "sayi" değişkeni de display'de gösterilecek sayının tümünü ifade etmektedir.

`output_low(display_1);`, `output_low(display_2);` komutları ile ilk başta birinci ve ikinci display'ler pasif hale getirilmiştir. Yani RA0 ve RA1 uçları bu komutla lojik-0 yapılmıştır. Dolayısıyla transistörlerin beyzine lojik-0 geleceğinden transistörler yalıtımda olacaktır ve ortak katot display'in katot uçlarına şase bağlantısı olmayacağından display'ler sönük olacaktır. `sayi=0;` komutu ile de gösterilecek sayının ilk değeri sıfır olarak atanmıştır.

`while(1)` komutu ile sonsuz bir döngü oluşturulmuştur. `birler=sayi%10;`, `onlar=sayi/10;` ve komutlarıyla display'de gösterilecek sayının birler ve onlar basamağı hesaplanmıştır. Birler basamağı hesaplanırken mod'lu bölme kullanılmıştır. Bu işlem "sayi" değişkeninin 10'a bölümünden kalanı verir. Bu da "sayi" değişkeninin birler basamağı demektir. "sayi" değişkeninin onlar basamağını hesaplamak içinse "sayi" değişkeni 10'a bölünmüştür. Bu işlem sonucundaki bölüm değerinin tam kısmı bize "sayi" değişkeninin onlar basamağını verir. Tam kısmı dendi çünkü "onlar" değişkeni tamsayı olarak tanımlanmıştır. Daha sonra ilk başta `output_high(display_2);` komutuyla RA1 pin'i (pin_a1) çıkışı lojik-1 yapılmış, dolayısıyla Q2 transistörü iletime geçirilmiştir. Q2 iletime geçince kolektör ve emitör arası kısa devre olunca, sağdaki display'in katot ucu şase bağlantısına kavuşmuş olacaktır. Yani sağdaki display aktif olmuş olacaktır. Aktif olan bu display'e `output_b(digit[birler]);` komutu ile gösterilecek sayının birler basamağı gönderilmiş ve ardından `delay_ms(5);` komutu ile de 5 msn süre gecikme verilmiştir. Ardından `output_low(display_2);` komutu ile de bu display pasif hale getirilmiştir. Hemen ardından `output_high(display_1);` komutu ile de anlatılan aynı mantıkla soldaki display aktif hale getirilmiş ve `output_b(digit[onlar]);` komutu ile de bu display'de sayının onlar basamağı gösterilmiştir. Yine belli bir süre gecikme verilmiş ve `output_low(display_1);` komutuyla pasif hale getirilmiştir. Bu şekilde belli bir süre aralıkla bir ilk display bir ikinci display aktif hale getirilerek gerekli bilgiler gönderilerek tarama yapılır.

`if (input(buton))` komutu ile "Arttır" butonuna basılıp basılmadığı test edilmektedir. Devre şemasında görüldüğü üzere butona basılmadığı sürece RA2 girişine lojik-0 bilgisi, butona basıldığında ise girişe lojik-0 bilgisi gelmektedir. Dolayısıyla "Arttır" butonuna basıldığında `input(buton)` komutu, 1 bilgisi ile geri dönecektir. Yani komut if (1) şeklini alacaktır ve şart (if komutunda parentez içinde belirtilen kontrol şartı) doğru kabul olmuş olacağından (parantez içindeki şart yanlış veya parantez içindeki işlem sonucu "0" olsaydı şart yanlış kabul edilecek ve if döngüsü komutları işlenmeden atlanacaktı) if döngüsü komutları işlenecektir. If döngüsü komutlarındaki `delay_ms(10);` ile verilen gecikme, butona basılınca meydana gelen arkları önlemek için verilmiştir. sayi++ komutu ile de her butona basılışta "sayi" değişkeni içeriğinin 1 artması sağlanmıştır. `if(sayi==100)` ve `sayi=0` komutlarıyla da "sayi" değişkeni içeriği 100 olunca değişken değerinin sıfırlanması sağlanmıştır. Böylece sayma işlemi 0-99 arasında olacak ve 99'dan sonra tekrar 0'a dönülecektir. Ardından tekrar "sayi" değişkeninin yeni değerinin birler ve onlar basamağı hesaplatılmıştır.

Butona her basışta "sayi" değerinde sadece bir artma olması için `while(input(buton))` komutu kullanılmıştır. Bu komutla butona basıldığında komut while(1) şeklini alacak ve while döngüsü komutu butondan el çekilene kadar işleyecektir. While döngüsü komutları programın ana fonksiyon (main ()) kısmının başlarında kullanılan display tarama komutları ile aynıdır. Böylece butona basıldığında "sayi" değeri 1 artacak ve butondan el çekilene kadar yeni "sayi" değeri display'de tarama işlemi ile gösterilecektir. Butondan el çekilince while(1) döngüsünün ilk başında olan tarama komutları sayesinde mevcut değer display'de görünür kalacaktır. Program çalışması bu şekilde devam edecektir.

Uygulama devrenizi PIC PROG/DEKA PIC16F87x deneme kartınızda denerken Port B Jumper'ını 7 seg konumuna almayı unutmayınız.

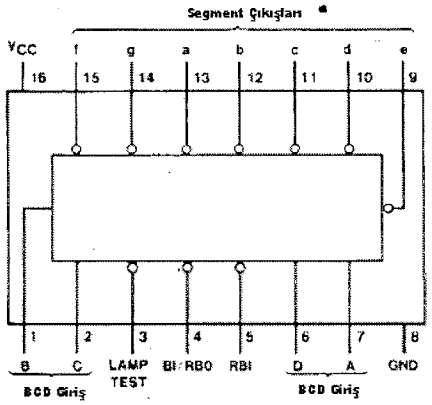
7.7. DISPLAY UYGULAMASI-3 (7448 ENTEGRESİ VE BUTON İLE YUKARI-AŞAĞI 0-99 SAYICI)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

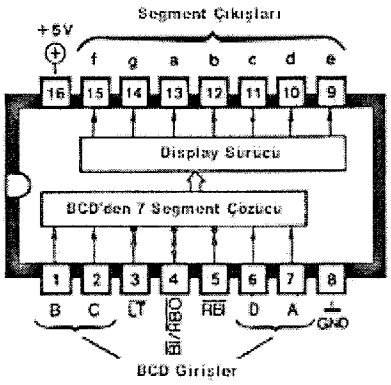
Önceki display uygulamalarında display'ler mikrodenetleyici tarafından direkt olarak sürüldü ve bu işlem için 7 adet veri için ve 2 adet display seçimi için toplam 9 çıkış ucu kullanıldı. Çoğu sistem tasarımlarında denetleyici çıkışları sistem için yeterli olmayabilir. Bu nedenle display sürme işlemi için BCD'den 7 segment display'e kod çözücü entegreleri de kullanılabilir. Bu sayede kullanılan uç sayısı azaltılmış olur. Ortak katot display için 7448 entegresi, ortak anot display için 7447 entegresi display sürücü entegresi olarak kullanılabilir. Bu entegre vasıtasıyla 4 bit ile bir display'e bilgi gönderilebilir. Böylece veri göndermek için 7 pin kullanmak yerine 4 pin kullanılır. Uygulama devresinde iki adet buton kullanılmıştır. Birisi arttır diğeri azalt butonudur. Display olarak da iki display'i içinde barındıran tümleşik ortak katot display kullanılmıştır. 7448 entegresinin 4 kontrol ucu, 8 de çıkış ucu vardır. Şekil-7.11'de 7447 entegresi, Şekil-7.12'de ise 7448 entegresi görülmektedir. Şekil-7.13'de ise 7447 ve 7448 entegrelerinin girişine verilen değerler sonucunda çıkışından alınan durumlar görülmektedir. Görüldüğü gibi hangi çıkış alınmak isteniyorsa direkt olarak o veri girişe 4 bit olarak verilmesi yeterlidir. Örneğin display'de 6 verisinin görülmesi isteniyorsa 7448'in giriş uçlarına 6 yani binary olarak 0110 değeri gönderilmesi yeterlidir. Çıkış uçları 220 Ω'luk dirençler ile display'e verilmiştir. B port'unun RB4 ve RB5 bit'leri display seçimi, A port'unun RA0 ve RA1 pin'leri ise butonlar için kullanılmıştır. Böylece sadece B port'unun 6 bit'i ile komple bir display sürme işlemi gerçekleştirilmiştir. Display seçim uçları NPN transistorleri ile kontrol edilmiştir. Uygulama devresinin şekli Şekil-7.12'de verilmiştir. 7447 entegresinin 8 numaralı bacağı V_{SS}, 16 numaralı bacağıda V_{DD} beslemesidir. Entegrenin A, B, C, D uçları BCD giriş uçları, a, b, c, d, e, f, g uçları ise display segment çıkış uçlarıdır. Entegredeki LT ucu display test, BI/RBO ve RBI uçları ise display karartma uçlarıdır. Bu üç uçta normal çalışmada pozitif besleme

gerilimine (V_{DD}) bağlanmalıdır. Eğer devrenizde ortak anot display yerine ortak katot display kullanmak isterseniz 7447 entegresi yerine 7448 entegresini kullanmanız gerekir.

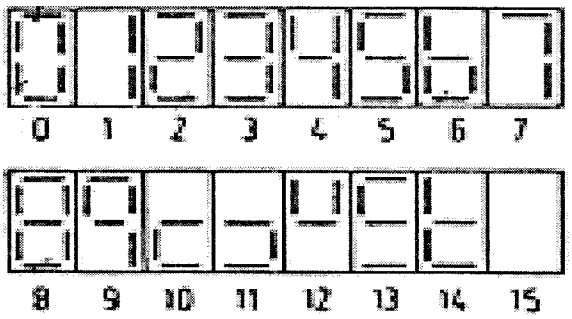
NOT= Devreyi proteus programında denediğinizde tam olarak simülasyon yapamayabilirsiniz. Devreyi proteus programında denerken display seçim uçlarını direkt olarak denetleyiciden display'e veriniz. Arada transistör kullanmayınız. Bu şekilde benzetim yaparken programda çıkışları terslemeyi unutmayınız (Normalde istenen display'in aktif olması için transistorün aktif olması gerekir. Bu nedenle display kontrol çıkışı lojik-1 yapılır. Böylece transistör iletime geçer ve display'in ortak katot ucu şase bağlantısı alır ve böylece istenen display seçilmiş olur. Fakat transistör kullanmadan proteus programında benzetim yapmak istenirse, istenen display'in aktif yapılması için kontrol çıkışı terslenerek lojik-0 yapılmalıdır).



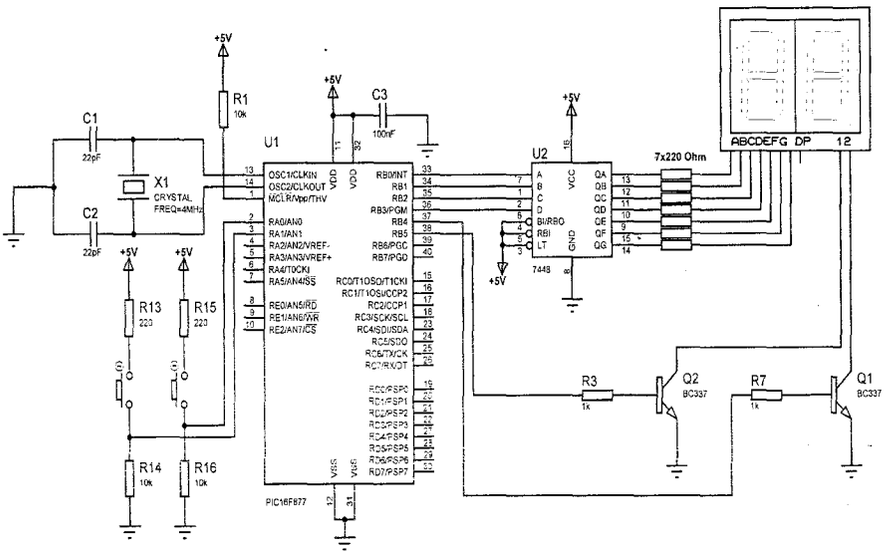
Şekil-7.11. 7447 entegresi pin diyagramı



Şekil-7.12. 7448 entegresi pin uçları



Şekil-7.13. 7447 entegresinde BCD girişlere göre çıkış görünümleri.



Şekil-7.14. *7448 ile yukarı/aşağı 0-99 sayıcı devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı        : Display Uygulaması-3(7448 Entegresi ve Buton
                       İle 0-99 Yukarı/Aşağı Sayıcı)
*****************************************************************/
#include <16f877.h>    // Kullanılacak denetleyicinin
                       //başlik dosyası tanıtılıyor.
//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor
#use fast_io(a) // Port yönlendirme komutları a port'u için geçerli
#use fast_io(b) // Port yönlendirme komutları b port'u için geçerli

#define arttir pin_a0     // arttir butonu pin_a0 ile eşleniyor
#define azalt  pin_a1     // azalt butonu pin_a1 ile eşleniyor

char birler=0,onlar=0,sayi=0;//char tipinde değişkenler tanımlanıyor

// Ortak katot display için veri değerleri
const int digit[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07,
0x7F, 0x6F};

/******** ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
```

```
setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
setup_adc(ADC_OFF);               // ADC birimi devre dışı
setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

set_tris_a(0x03); // RA0 ve RA1 giriş olarak yönlendiriliyor
set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

while(1)      // Sonsuz döngü
{
   output_b(digit[birler]|(0b00010000)); // 1.display
   //aktif, 2.display pasif yapılıyor ve birler basamağı
   bilgisi B port'una gönderiliyor

   delay_ms(5);  // Gecikme veriliyor
   output_b(digit[onlar]|(0b00100000)); //1.display pasif,
   2.display aktif yapılıyor ve birler basamağı bilgisi B
   port'una gönderiliyor

   delay_ms(5); // Gecikme veriliyor

  // arttır butonuna basıldığında döngüyü işle
   if (input(arttir))
   {
      sayi++;  //sayi değerini 1 arttır. sayi=sayi+1 anlamında

      birler=sayi%10;          // birler hanesi hesaplanıyor
      onlar=sayi/10;           // onlar hanesi hesaplanıyor

      while(input(arttir)) //arttır butonundan el çekilene kadar
                           //bekle
      {
         output_b(digit[birler]|(0b00010000)); // 1.display
         //aktif, 2.display pasif yapılıyor ve birler basamağı
         bilgisi B port'una gönderiliyor

         delay_ms(5);  // Gecikme veriliyor
         output_b(digit[onlar]|(0b00100000)); //1.display pasif,
         2.display aktif yapılıyor ve birler basamağı bilgisi B
         port'una gönderiliyor

         delay_ms(5); // Gecikme veriliyor
      }
   }

   /* azalt butonuna basıldığında ve sayi değişkeni 0 olmadığı
      sürece döngüyü işle*/
  if (input(azalt)&&(sayi!=0))
      {
      sayi--;     // sayi değerini 1 azalt
```

```
            birler=sayi%10;          // birler hanesi hesaplanıyor
            onlar=sayi/10;           // onlar hanesi hesaplanıyor

            while(input(azalt))//azalt butonundan el çekilene kadar
                               //bekle
            {
               output_b(digit[birler]|(0b00010000)); // 1.display
               //aktif, 2.display pasif yapılıyor ve birler basamağı
               bilgisi B port'una gönderiliyor

               delay_ms(5); // Gecikme veriliyor
               output_b(digit[onlar]|(0b00100000)); //1.display pasif,
               2.display aktif yapılıyor ve birler basamağı bilgisi B
               port'una gönderiliyor
               delay_ms(5); // Gecikme veriliyor
            }
         }
         if (sayi==100)   // sayi değeri 100 ise değeri sıfırla
            sayi=0;
      }
}
//*****************************************************************
```

Programda önceki uygulamalarda olduğu gibi genel tanımlamalar yapıldıktan sonra programda kolaylık olsun diye "arttır" ve "azalt" buton girişleri #define komutu ile sabit olarak tanımlanmıştır. Bu komutlardan sonra derleyici program "arttir" kelimesini gördüğü yere "pin_a0", "azalt" kelimesini gördüğü yere "pin_a1" yazacaktır. Programda kullanılan değişken ve sabitler tanımlandıktan sonra ana fonksiyona geçilmiştir. Ana fonksiyonda kullanılan port'lar devrede giriş/çıkış durumlarına göre yönlendirilmiştir. Gösterilecek sayi değişkeninin birler ve onlar basamakları display uygulaması-2'de olduğu gibi aynı şekilde hesaplanmıştır.

`output_b(digit[birler]|(0b00010000));` komutu ile "sayi" değişkeninin birler basamağı B port'una gönderilmektedir. Fakat bu komutla birler basamağı aynı zamanda binary olarak `0b00010000` verisi ile OR işlemine tabi tutulduktan sonra B port'una gönderilmektedir. Bunun sebebi "sayi" değişkeninin birler basamağı değeri B port'una gönderilirken aynı zamanda da birinci display'in aktif, ikinci display'in pasif yapılmasıdır. Dikkat edilirse OR işleminde sadece 5. ve 6. bit'ler 1 veya 0 yapılmaktadır. Bilginin 5. ve 6. bit'leri RB4 ve RB5 bit'lerine tekâmül etmektedir. Bu bit'lerde display seçim uçlarıdır. Diğer bit'ler hep 0'dır. Böylece OR'lama işleminde sadece display seçim bit'leri olan RB4 ve RB5 bit'lerinde oynama yapılmıştır. Diğer bit'ler hep 0 ile OR'lanacağından değerlerinde herhangi bir değişim olmayacaktır. Bu komuttan sonra display'de gösterilen sayının belli bir süre yanık kalması için `delay_ms(5);` komutu ile gecikme verilmiştir. Ardından `output_b(digit[onlar]|(0b00100000));` komutu ile de birinci display pasif, ikinci display ise aktif yapılmış ve "sayi" değişkeninin onlar basamağı B port'una gönderilmiştir. Dikkat edilirde bu komuttaki OR'lama işleminde OR'lama yapılan sayının 4.bit'i 0'dır, yani RB4 çıkışı lojik-0'dır. Dolayısıyla birinci display pasiftir.

Sayının 5.bit'i ise 1'dir, yani RB5 çıkışı lojik-1'dir. Dolayısıyla ikinci display aktiftir. Bu komuttan sonra da yine 5 msn gecikme verilmiştir. Başlangıçta "birler" ve "onlar" değişkenlerinin içeriği sıfır olduğundan display'lerde 00 rakamları görünecektir.

`if (input(arttir))` komutu ile "arttir" butonuna basılıp basılmadığı kontrol edilmektedir. Devre şemasında görüldüğü gibi butona basılmadığında girişe lojik-0 bilgisi, butona basıldığında ise girişe lojik-1 bilgisi gelmektedir. Arttır butonuna basıldığında `input(arttir)` komutu 1 değeri ile geri döner. Dolayısıyla komut if (1) şeklini alır, şart doğru olmuş olur ve if döngüsü işlemeye başlar. Döngü komutlarında ilk başta `sayi++;` komutuyla sayi değişkeni içeriği bir arttırılmıştır. sayi++ komutu sayi=sayi+1; komutu ile aynı anlamdadır. Bu komuttan sonra kullanılan `birler=sayi%10;` ve `onlar=sayi/10;` komutlarıyla display'de gösterilecek sayının birler ve onlar basamağı hesaplanmıştır. `while(input(arttir))` komutuyla da basılan butondan el çekilene kadar display'ler de o anki sayı değerinin görülmesi sağlanmıştır. Arttır butonuna bir kere basıldığında "sayi" değişkeni değeri bir artar ve display'ler de görünür. Buton basılı kaldığı sürece display'deki görünen sayı bir daha artmaz. Bu sayede butondan elimizi çekmesek dahi artış sadece bir kere olacaktır. Butondan elimizi çektiğimizde girişe lojik-0 bilgisi gelecektir. If komutu if (0) şeklini alacak yani şart yanlış olmuş olacağından döngüden çıkılacaktır.

`if (input(azalt)&&(sayi!=0))` komutuyla da Azalt butonuna basılıp basılmadığı kontrol edilmektedir. Bu döngünün işleme girmesi için hem azalt butonuna basılmalı hem de "sayi" değişkeni içeriği sıfır olmamalıdır. Dikkat edilirse bu iş için komutta Mantıksal AND operatörü kullanılmıştır. "sayi" değişkeninin sıfıra eşit olmama şartı eğer "sayi" değişkeni sıfırsa "sayi" değişkeni azaltılıp eksi değer almasın diyedir. Azalt butonuna basıldığında `sayi--;` komutuyla "sayi" değişkeni içeriği bir azaltılmıştır. sayi--; komutu sayi=sayi-1 komutu ile aynı anlamdadır. Azalt butonu döngüsünde de arrtir döngüsünde olduğu gibi `while(input(azalt))` komutu ile buton basılı kaldığı sürece başka azaltma işlemi yapılmaması sağlanmıştır. Döngüdeki diğer işlemler arttır butonu döngüsü ile aynıdır.

En son olarak da `if (sayi==99)` komutu ile "sayi" değişkeninin değerinin 99'a ulaşıp ulaşmadığı kontrol edilmekte. Eğer "sayi" değeri 99 ise değişken içeriği sıfırlanmaktadır. Program bu şekilde çalışmaya devam eder.

NOT= Devreyi Proteus benzetim programında denemek isterseniz tarama işlemini tam olarak göremeyebilirsiniz. Benzetim işleminde tarama işlemini görmek için display kontrol uçlarını transistör ile değil, direk bağlayınız. Display seçme uçlarını direkt bağlayacağınız için programda display seçme bit'lerini terslemeyi unutmayın (Transistör bağlantısı varken RB4 ve RB5 display seçme çıkışları lojik-1 ise display seçilmiş olur. Direkt bağlamada display'i seçmek için bu çıkışlar lojik-0 yapılmalıdır.).

8. Bölüm

Port Çoğullama ve Matris LED Display Uygulamaları

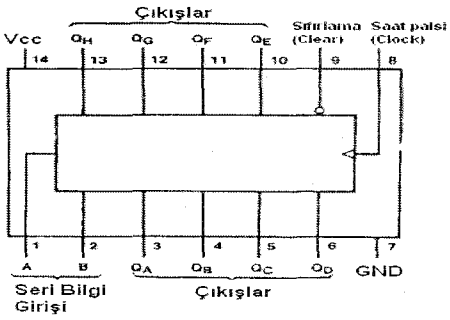
- 74164 Entegresi
- 74138 Entegresi
- Entegresi
- 74164 İle 2 Çıkışla 7 Segment Display Sürme
- 74138 İle Port Çoğullama Uygulaması
- 8255 İle Port Çoğullama Uygulaması
- Matris LED Display
- Matris LED Display Uygulaması

PIC16F877 denetleyicisi çoğu uygulamalar için yeterli sayı da I/O pin sayısına sahiptir. Fakat bazı karmaşık ve farklı sistem modüllerini içinde barındıran uygulamalarda mevcut pin sayısı yeterli olmayabilir. Bazen tasarladığınız sistemlerinizde basit bir kontrol işlemi için çok sayıda çıkış ihtiyacınız olabilir. Bu gibi sistemlerinizde, ihtiyacınız olmayan değişik çevresel birimlerini (ADC, LCD, I^2C, SPI, PSP vb.) içinde barındıran pahalı denetleyiciler yerine çevresel birim içermeyen ucuz denetleyicileri kullanmak daha ekonomiktir. Pahalı denetleyiciler yerine daha ucuz port çoğullama çözümleri kullanılabilir. Bu bölümde kontrol işlemlerinin normalden daha az pin sayısı ile gerçekleştirilmesine olanak sağlayan bazı entegreler üzerinde durulmuştur. Kitapta incelenen örnek uygulamalarından birden fazlasını içinde barındıran karmaşık sistemler tasarlamak istediğinizde incelenen port çoğullama çözümlerini kullanabilirsiniz. Ayrıca bu bölümde Matris LED display'ler incelenmiş ve bu tarz display'leri sürme yöntemleri anlatılmıştır.

8.1. 74164 ENTEGRESİ

74164 entegresi popüler bir kaymalı kaydedici (shift register) entegresidir. Bu entegre 8 bit'lik bir bilgiyi seri giriş hattından alır ve 8 bit'lik paralel çıkış hattına gönderir. Şekil-8.1'de görüldüğü gibi 74164 entegresi 2 adet 1 bit'lik seri giriş

ucuna, saat (clock) palsi ucuna, temizleme (clear) ucuna ve 8 bit'lik çıkış uçlarına sahiptir. Piyasada bulunan 74164 entegreleri saat frekansı genellikle 36 Mhz olmasına rağmen daha yüksek saat frekanslarında çalışan 74164 entegreleri de vardır. Örneğin DM74164 ve SN74164 kodlu entegreler maksimum 36 Mhz saat frekansına sahipken, MM74HC164 kodlu entegre daha gelişmiştir ve saat frekansı olarak 50 Mhz'e kadar çalışabiimektedir. Tablo-8.1'de 74164 entegresinin fonksiyon tablosu verilmiştir. Tabloda "H" harfi lojik-1'i (**H**igh), "L" harfi lojik-0'ı (**L**ow), "X"harfi ise bahsedilen girişin lojik-0 veya lojik-1 olmasının fark etmeyeceğini ve ↑ şekli ise lojik-0'dan lojik-1'e geçişi ifade etmektedir.

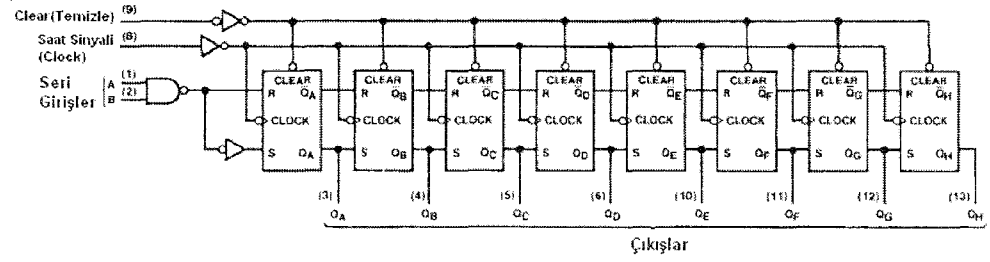


Şekil-8.1. *74164 pin diyagramı.*

Girişler				Çıkışlar			
Clear	Clock	A	B	Q_A	Q_B	...	Q_H
L	X	X	X	L	L	...	L
H	L	X	X	Q_{A0}	Q_{B0}	...	Q_{H0}
H	↑	H	H	H	Q_{An}	...	Q_{Gn}
H	↑	L	X	L	Q_{An}	...	Q_{Gn}
H	↑	X	L	L	Q_{An}	...	Q_{Gn}

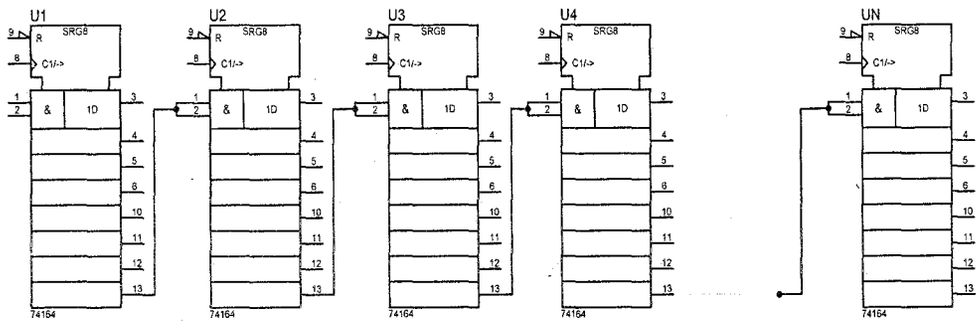
Tablo-8.1. *74164 fonksiyon tablosu.*

Vcc = Entegre pozitif besleme ucu.
GND = Entegre şase ucu.
Clear = Bu uçla tüm çıkışlar temizlenir yani sıfırlanır. Clear ucu lojik-0'da (Low) aktiftir. Bu nedenle bu uca normal çalışmada lojik-1 seviyesi uygulanmalıdır.
Clock = Saat palsi giriş ucu. Sinyalin düşük (Low - lojik-0) seviyesinden yüksek (High - lojik-1) seviyesine geçişte bu uç aktiftir.
A, B = Seri giriş uçlarıdır.
QA...QH = Paralel çıkış uçlarıdır.



Şekil-8.2. *DM74164 entegresi iç yapısı.*

74164 entegresinin çalışmasını daha iyi anlamak için Şekil-8.2'de verilen entegre iç yapısı incelenmelidir. 74164 entegreleri 8 adet flip-flop içerirler. Değişik firmaların üretimi olan 74164 entegreleri değişik flip-flop dizaynına (R-S, D vb.) sahiptir. Seri girişten uygulanan ilk bilgi saat sinyalinin düşük seviyeden yüksek seviyeye geçişinde ilk flip-flop çıkışında (QA) görünür. Saat sinyalinin ikinci palsinde bu sefer seri girişteki bilgi yine QA flip-flop'una kaydedilir ve QA çıkışında görünür. İlk seri bilgi ise bu sefer bir bit kayarak QB flip-flop çıkışında görünür. Bu işlemler bu şekilde devam eder. Böylece seri girişten gelen bilgi her saat sinyali ile bir bit sağa kayarak çıkışa aktarılmaktadır. Böylece 8 bit'lik bir seri bilginin en son girilen bit'i QA çıkışında, ilk girilen bit'i ise QH kaydedicisinde görünür. 74164 entegreleri birbirlerine bağlanarak daha fazla sayıda çıkış ucu elde edilebilir. Örneğin kayan yazı uygulamalarında Şekil-8.3'deki gibi birden fazla 74164 entegresini birbirine bağlarken ilk entegrenin QH çıkışı (13 nolu pin), diğer entegrenin seri girişine (1,2 nolu pin'ler) bağlanır. Bu şekilde istenen port sayısı için gerekli kadar 74164 entegresi birbirine bağlanabilir. Şekilde görünen 74164 sembolü ANSI/IEEE standardına göre 74164 entegresinin lojik sembolüdür.

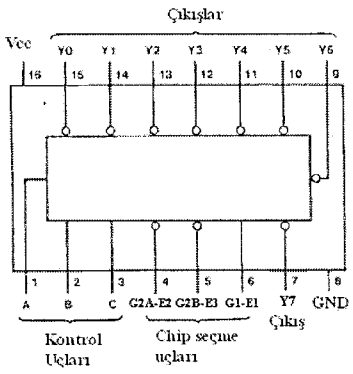


Şekil-8.3. *Birden fazla 74164'ün birbirine bağlanması.*

8.2. 74138 ENTEGRESİ

73138 entegresi kod çözücü/çoğullama (decoder/demultiplexer) entegresidir. Bu entegre ile 3 bit'lik bir kontrol bilgisi ile 8 bit'lik çıkış bilgisini kontrol edebiliriz. Böylece 8 bit'lik bir kontrol işlemini 3 bit ile gerçekleştirerek kullanılacak pin sayısını azaltmış oluruz yani port çoğullama gerçekleşmiş olur. Ayrıca 74138 entegresi 3 adet entegre (yonga - chip) seçme uçlarına sahiptir. Bu entegre seçme bit'lerinin biri lojik-0'da aktif, biri ise lojik-1'de aktiftir. Böylece entegre seçme uçları istenen her durum için harici değil kapısı kullanmaya gerek bırakmaz. Şekil-8.4'te 74138 entegresinin pin diyagramı, Tablo-8.2'de de fonksiyon tablosu görülmektedir. 74138 entegreleri birbirine bağlanarak istenen sayıda çıkış sayısı elde edilebilir. Çoklu kullanımlarda 74138 entegresinden biri diğer 74138 entegrelerini kontrol etmek için kullanılabilir.

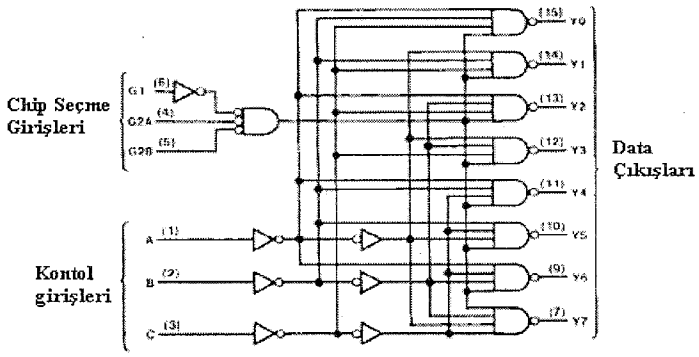
A, B, C (A0, A1, A2)= Kontrol uçlarıdır. Hangi çıkışın seçileceği bu bit'ler ile belirlenir. Bazı kaynaklarda bu uçlar A0, A1 ve A2 şeklinde gösterilmektedir.
G2A, G2B, G1 = Entegre (chip) seçme bit'leridir. G2A ve G2B lojik-0'da aktif, G1 lojik-1'de aktiftir. Fonksiyon tablosunda görüldüğü gibi bir entegrenin seçilmiş olabilmesi için G1 ucunun lojik-1, G2 yani G2A+G2B uçlarının toplamı lojik-0 olmalıdır. Bazı kaynaklarda bu uçlar $\overline{E2}, \overline{E3}, E1$ şeklinde gösterilmektedir.
Y0...Y7 = Çıkış uçlarıdır. Çıkış uçları terslenmiş haldedir.



Şekil-8.4. 74138 Pin diyagramı.

Girişler					Çıkışlar							
Chip seçme		Kontrol										
G1	G2(G2A+G2B)	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

Tablo-8.2. 74138 Fonksiyon tablosu.

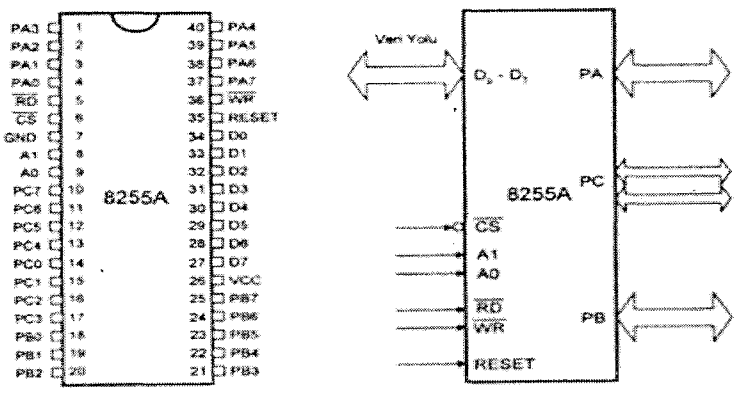


Şekil-8.5. 74138 Lojik diyagramı.

Şekil-8.5'de görüldüğü gibi entegre seçme uçlarında G2 (G2A ve G2B) girişi değillenmiştir. G1 girişi ise iki kere değillendiğinden mantıksal olarak değillenmemiştir. Ayrıca bu uçlar (G1, G2A, G2B) AND kapısı ile VE işlemine tabi tutulmuştur. Dolayısıyla devrenizde kullandığınız 74138 entegresini seçmek için G1 ucu mutlaka lojik-1, G2A, G2B girişlerinin mantıksal toplamı (G2=G2A+G2B) yani G2 lojik-0 olmalıdır. Bu şart sağlandıktan sonra A, B, C uçlarından istenen çıkış için değer girilir. Örneğin bu uçlara 4 değeri (C=1, B=0, A=0) gönderilirse Y4 çıkışı lojik-0, diğer çıkışlar lojik-1 olur. Bu mantıkla istenen yonga ve çıkış kontrol edilir.

8.3. 8255 ENTEGRESİ

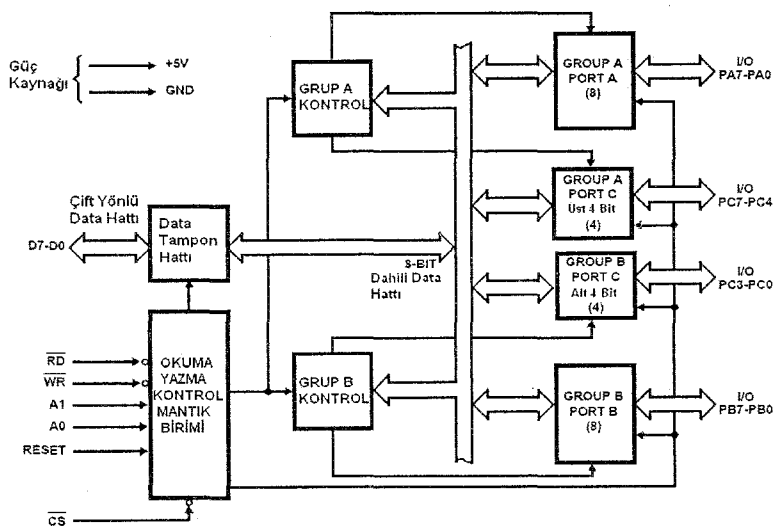
CMOS yapıda programlanabilir çevresel ara yüz elemanıdır. 8255, çoğullama için kullanılan diğer entegrelere oranla daha karmaşık ve sağlam bir yapıya sahiptir. 8255'i diğer entegrelerden ayıran en önemli özelliklerden biri programlanabilmesidir. 8255 24 adet TTL uyumlu programlanabilir giriş/çıkış (I/O) ucuna sahiptir. Şekil-8.5'de 8255 entegresinin pin diyagramı görülmektedir.



Şekil-8.5. *8255 Entegresi pin diyagramı.*

Pin İsmi	Tipi	Açıklama
Vcc		5V ucu. Vcc ile GND arasında 0.1 µF kondansatör bağlanmalı
GND		Şase
D0-D7	I/O	Data hattı
Reset	I	Sıfırlama ucu.
$\overline{CS}$	I	Entegre seçme ucu
$\overline{RD}$	I	Okuma kontrol ucu
$\overline{WR}$	I	Yazma kontrol ucu
A0, A1	I	Port veya kontrol kaydedicisi seçme bit'leri
PA0-PA7	I/O	8 bit'lik A Port'u
PB0-PB7	I/O	8 bit'lik B Port'u
PC0-PC7	I/O	8 bit'lik C Port'u

Tablo-8.3. *8255 pin açıklamaları.*



Şekil-8.6. *8255 Fonksiyonel blok diyagramı.*

Data Tampon Hattı (Data Bus Buffer)= Tampon hattı 3 durumlu iki yönlü 8 bit'lik data ara yüz hattıdır. Data bit'leri bu hat içinden alınır veya gönderilir. 8255 kontrol bilgileri (control word) ve durum bilgileri (status information) de bu hat üzerinden iletilir.

Okuma/Yazma Kontrol Mantık Birimi (Read/Write and Control Logic) = Tüm iç ve dış data, kontrol ve durum bilgilerinin yönetildiği birimdir. Bu birime adres ve kontrol giriş uçları ile erişilir.

$\overline{CS}$ = Entegre seçme ucu. Lojik-0'da aktiftir.

$\overline{RD}$ = Okuma kontrol ucu. Lojik-0'da aktiftir. İşlemcinin 8255'den okuma yapmasını sağlar.

$\overline{WR}$ = Yazma kontrol ucu. Lojik-0'da aktiftir. İşlemcinin 8255'e data ve kontrol bilgilerini yazmasına izin veir.

A0 ve A1 = Bu girişler RD ve WR giriş değerleri ile birlikte port kaydedicilerini veya kontrol kaydedicisini seçer. Tablo-8.4'te kaydedicileri seçmek için gerekli giriş değerleri verilmiştir.

Tablodan görüleceği üzere örneğin okuma girişi (RD) ve entegre seçme girişi (CS) aktif olduğunda okuma işlemi seçilmiş olur. Bu durumda A1=0, A0=0 ise Port A'dan okuma işlemi, A1=1, A0=0 ise Port C'den okuma işlemi yapılır. A1=1, A0=1 ise kontrol kaydedicisi bilgileri okunur. Yazma girişi (WR) ve entegre seçme bit'i (CS) aktif olduğunda ise seçilen port'ta veya konrol kaydedicisine yazma işlemi yapılır. Örneğin yazma durumunda A1=0 A0=1 ise B port'una bilgi yazılır. Entegre seçme bit'i aktif değilken veya hem okuma hem de yazma bit'i aktifken entegrede geçersiz durum oluşur.

A1	A0	RD	WR	CS	Okuma İşlemleri
0	0	0	1	0	Port A
0	1	0	1	0	Port B
1	0	0	1	0	Port C
1	1	0	1	0	Kontrol Kaydedicisi
A1	**A0**	**RD**	**WR**	**CS**	**Yazma İşlemleri**
0	0	1	0	0	Port A
0	1	1	0	0	Port B
1	0	1	0	0	Port C
1	1	1	0	0	Kontrol Kaydedicisi
					Geçersiz Durumlar
X	X	X	X	1	
X	X	1	1	0	

Tablo-8.4. *8255 Okuma/Yazma işlemleri.*

Reset (Sıfırlama)= Reset girişi lojik-1'de aktiftir. Bu giriş aktif olduğunda tüm kontrol kaydedicilerini siler ve tüm port'ları (A, B, C) giriş moduna kurar.

Grup A ve Grup B Kontrolleri
8255'de port yönlendirme işlemleri için iki adet kontrol birimi vardır. Bunlar Grup A ve Grup B'dir. Her kontrol grubu A, B, C port'larının belli bir kısmını kapsar.

Grup A Kontrol → A Port'u ve C port'unun yüksek değerlikli 4 bit'i (C7-C4)
Grup B Kontrol → B Port'u ve C port'unun düşük değerlikli 4 bit'i (C0-C3)

Port A, Port B ve Port C

A Port'u, 8 bit data çıkış latch (mandal)/tampon (buffer) ile 8 bit data giriş latch'ine sahiptir.

B Port'u, 8 bit data giriş/çıkış latch/tampon ile 8 bit data giriş tamponuna sahiptir.

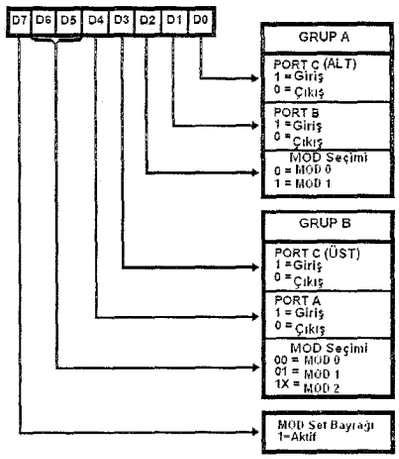
C Port'u, 8 bit data çıkış latch/tampon ile 8 bit data giriş latch'ine sahiptir. C port'u diğer port'lardan ayrı olarak ikiye bölünmüştür. Çalışma modu kontrolü seçimi ile C port'unun yüksek değerlikli 4 bit'i ile düşük değerlikli 4 bit'i ayrı ayrı giriş ve/veya çıkış olarak yönlendirilebilir. Diğer port'lar bölünmediği için, diğer port'lar ya komple giriş yada komple çıkış olarak yönlendirilebilir.

8.3.1. 8255 Çalışma Modları ve Kontrol Kaydedicisi

8255, hangi port'ların giriş hangilerinin çıkış olacağını ve port çalışma şeklini belirlemek için 3 adet çalışma moduna sahiptir. Bunlar **Mod0, Mod1** ve **Mod2** isminde tanımlanmıştır.

Mod 0 → Basit Giriş/Çıkış Modu
Mod 1 → El sıkışmalı (Handshake) Giriş/Çıkış Modu
Mod 2 → Çift yönlü (Bidirectional) Giriş/Çıkış Modu

Mod 0'da bütün port'lar basit I/O port'ları olarak kullanılır. **Mod 1'de** Port A ve Port B, Port C' nin bit'lerini karşılıklı haberleşme sinyalleri olarak kullanır. **Mod 2** de Port A, Port C den karşılıklı haberleşme sinyalleri kullanarak iki yönlü veri transferi yapacak şekilde, Port B de Mod 0 veya Mod 1 de çalışmaya programlanabilir. Temel port çoğullama işlemlerinde genellikle Mod 0 çalışma şekli kullanılır. Bu nedenle biz Mod 0 çalışma modu üzerinde duracağız. 8255'de tüm ayarlar kontrol kaydedicisine yüklenen kontrol kelimesi (control word) bilgisi ile belirlenir. Şekil-8.7'de kontrol kelimesi ve port'ları yönlendirmek için gerekli bit durumları görülmektedir.



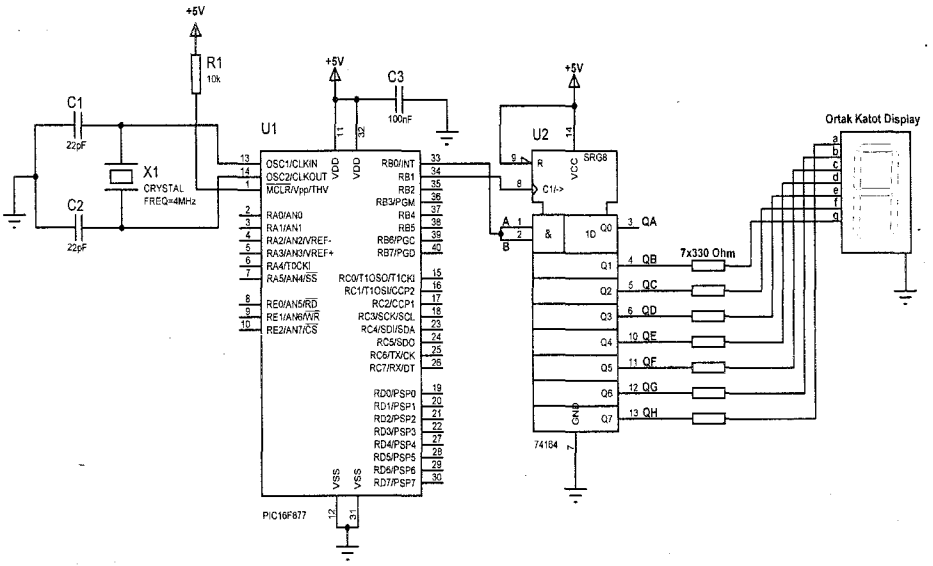
Şekil-8.7. *Kontrol kelimesi bit tanımlamaları.*

Kontrol kelimesinde 7. bit lojik-1 olmalıdır. Örneğin Mod 0 çalışma da A port'unun giriş, B port'unun çıkış ve C port'unun yüksek değerlikli 4 bit'inin giriş (ÜST), düşük değerlikli 4 bit'inin (ALT) çıkış olması isteniyorsa kontrol kelimesi 0b10011000 olmalıdır. Eğer A ve B port'unun giriş C port'unun ise komple çıkış yapılması isteniyorsa kontrol bilgisi 0b10010010 olmalıdır.

8.4. 74164 İLE 2 ÇIKIŞLA 7 SEGMENT DİSPLAY SÜRME

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

7. bölümde 2 çeşit display sürme yöntemi incelenmişti. Bunlardan ilki mikrodenetleyici ile direkt sürme yöntemiydi. Bu yöntemde display'e bilgi göndermek için 7 adet çıkış ucuna ihtiyaç vardı. İncelenen diğer yöntem ise display sürücü entegresi ile yapılan uygulamaydı. Bu yöntemde de 4 adet bilgi ucuna ihtiyaç vardı. Şimdi ise 74164 kaymalı kaydedici ile yapılan ve sadece 2 çıkış ucu kullanılarak 7 segment bir display'i sürme uygulaması incelenmiştir.



Şekil-8.8. *74164 ile 2 çıkış ucuyla 7 segment display sürme devre şeması.*

Devre Programı;

```
/*****************************************************************
 Uygulama Adı       : 74164 ile 2 Çıkışla 7 Segment Display Sürme
*****************************************************************/

#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         //dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor
#use fast_io(b) // Port yönlendirme komutları b port'u için geçerli

#define data pin_b0   // data sabiti pin_b0'a eşitleniyor
#define clock pin_b1  // clock sabiti pin_b1'e eşitleniyor

// Ortak katot display için veri değerleri
int digit[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7C,0x07,0x7F,0x6F};

char i=0, j=0; // karakter tipinde değişken tanımlaması yapılıyor

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
```

```
  setup_timer_1(T1_DISABLED);      // T1 birimi devre dışı
  setup_timer_2(T2_DISABLED,0,1);  // T2 birimi devre dışı
  setup_adc_ports(NO_ANALOGS);     // ANALOG giriş yok
  setup_adc(ADC_OFF);              // ADC birimi devre dışı
  setup_CCP1(CCP_OFF);             // CCP1 birimi devre dışı
  setup_CCP2(CCP_OFF);             // CCP2 birimi devre dışı

  set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

  output_b(0x00); // İlk anda B port'u çıkışı sıfırlanıyor

  while(1)  // Sonsuz döngü oluşturuluyor
  {
    for(i=0;i<10;i++)//Display'de gösterilecek 0-9 sayıları için

    {
       for(j=0;j<8;j++) // Gönderilecek verinin bit bit
                        //gönderilmesi için oluşturulan döngü
       {
          output_bit(data,bit_test(digit[i],j)); // Bit değeri
                                                 //data pin'ine gönderiliyor
          output_high(clock); // Clock sinyali lojik-1 yapılıyor
          output_low(clock);  // Clock sinyali lojik-0 yapılıyor
       }
       delay_ms(500); // Gecikme veriliyor
    }
  }
}
//*****************************************************************
```

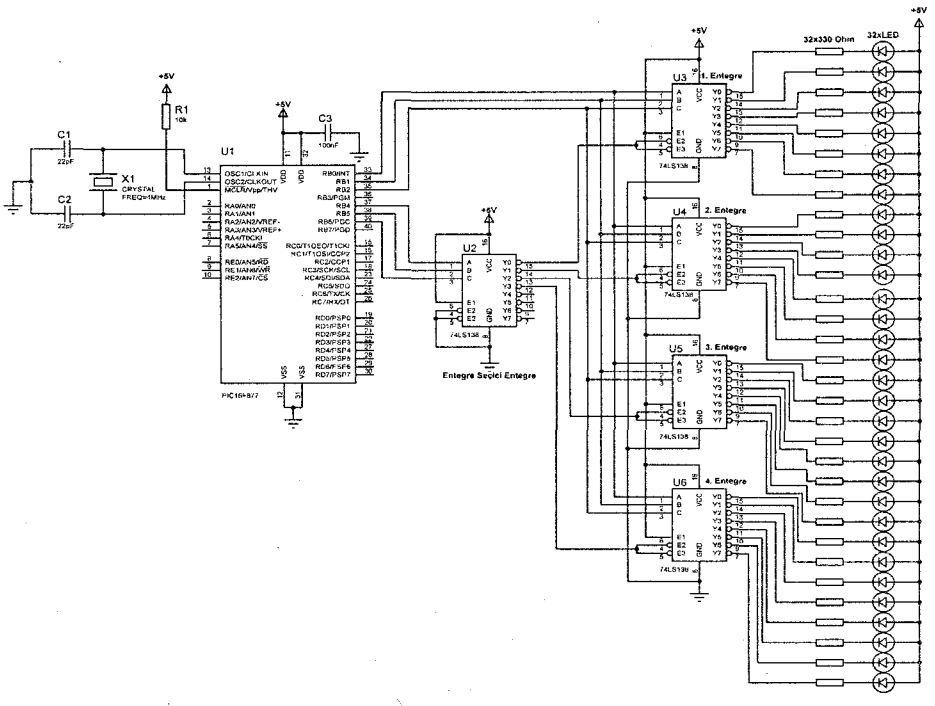
Programda genel tanımlama ve ayar komutları yazıldıktan sonra 74164 entegresinin data ucunu temsil eden "data" sabiti ile clock girişini temsil eden "clock" sabitleri ilgili pin'lerin yerine geçecek şekilde tanımlanmıştır. Programdaki ilk `for(i=0;i<10;i++)` döngüsü komutu ile display'de gösterilecek sayi değerleri veri bilgilerinin sırayla gönderilmesi sağlanmıştır. Döngüde i değişkeni sıfırdan başlayacak ve her döngü başa geldiğinde bir arttırılarak i değişkeni 10'dan küçük olana kadar döngünün dönmesi devam edecektir. Bu döngü ile 0-9 arası tüm sayıların değerleri display'e gönderilmiş olacaktır. Aynı döngü içinde tanımlanan ikinci `for(j=0;j<8;j++)` döngüsü ile de display'e gönderilecek sayi değerlerinin bit bit gönderilmesi sağlanmıştır. j değeri 0'dan başlayacak ve her döngü de j değeri 1 arttırılarak j değeri 8'den küçük olana kadar döngü dönecektir. Döngüde `output_bit(data,bit_test(digit[i],j));` komutu içinde bulunan `bit_test(digit[i],j)` komutu ile `digit[i]` dizi elemanının j. bit'inin değeri öğrenilmiştir. Daha sonra `output_bit(data,bit_test(digit[i],j));` komutu ile öğrenilen bu bit değeri "data" sabitine yani `pin_b0` çıkışına gönderilmektedir. Ardından `output_high(clock);` ve `output_low(clock);` komutları ile `clock` çıkışı yani `pin_b1` çıkışı yüksek seviyeden düşük seviyeye getirilmiştir. Böylece 74164 entegresinin clock sinyali verilmiş olur. Yani bu döngü

ile gönderilecek 8 bit'lik bilginin sırayla bit bit 74174'e gönderilmesi sağlanmıştır. İlk for döngüsü sonunda tüm sayı değerleri (0'dan 9'a) gönderilmiş olacaktır.

8.5. 74138 İLE PORT ÇOĞULLAMA UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

74138 entegresi ile ilgili teorik bilgiden sonra bu örnekte pratik olarak kullanımı incelenmiştir. Uygulama devresinde 5 adet 74138 entegresi ile PIC16F877'nin sadece 6 ucu kullanılarak 32 adet çıkış elde edilmiştir. Çıkışlara LED bağlanarak sırayla aktif olan çıkışın durumunun görsel olarak görülmesi sağlanmıştır. Devrede kullanılan 5 adet 74138 entegresinin 4 tanesi çıkış için, 1 tanesi ise çıkış için kullanılan 4 entegreyi seçmek için kullanılmıştır. Seçici olarak kullanılan 74138 entegresine 4 tane 74138 bağlanmıştır. İstenirse aynı entegreye bir 4 tane daha entegre bağlanarak çıkış sayısı 64'e çıkarılabilir. 74138 entegresinin çıkışları önceden anlatıldığı gibi değillenmiş durumdadır. Bu nedenle çıkışlara bağlanan LED'lerin anotları birleştirilerek +5V'a bağlanmıştır. Her LED'in katotu ise ilgili çıkışa bağlanmıştır. Devrede RB0, RB1 ve RB2 uçları çoğullama entegrelerinin çıkış uç numaralarını seçmek için kullanılmıştır. RB4, RB5 ve RB6 uçları ise devredeki 4 adet çoğullama entegresini aktif etmek (seçmek) için kullanılmıştır.



Şekil-8.9. *74138 ile port çoğullama uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı       : 74138 İle Port Çoğullama Uygulaması
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
osilatör frekansı belirtiliyor
#use fast_io(b) // Port yönlendirme komutları b port'u için geçerli

int i; // Tamsayı tipinde değişken tanımlanıyor

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor

   while(1) // Sonsuz döngü
   {
      for(i=0;i<8;i++) // 1. 74138 entegresi çıkış kontrolü
      {
         output_b(i|0b00000000); // İlk 74138 seçiliyor ve sırayla
                                 //  çıkışlar aktif yapılıyor
         delay_ms(250);          // Gecikme veriliyor
      }

      for (i=0;i<8;i++) // 2. 74138 entegresi çıkış kontrolü
      {
         output_b(i|0b00010000); // İkinci 74138 seçiliyor ve
                                 // sırayla çıkışlar aktif yapılıyor
         delay_ms(250);          // Gecikme veriliyor
      }

      for(i=0;i<8;i++) // 3. 74138 entegresi çıkış kontrolü
      {
         output_b(i|0b00100000); // Üçüncü 74138 seçiliyor ve
                                 // sırayla çıkışlar aktif yapılıyor
         delay_ms(250);          // Gecikme veriliyor
      }
```

```
        for (i=0;i<8;i++) //  4. 74138 entegresi çıkış kontrolü
        {
           output_b(i|0b00110000); // Dördüncü 74138 seçiliyor ve
                                   // sırayla çıkışlar aktif yapılıyor
           delay_ms(250);          // Gecikme veriliyor
        }
    }
}
//*****************************************************************
```

Programda ilk başta `for(i=0;i<8;i++)` döngüsü ile birinci 74138 entegresinin çıkış uçlarının kontrolü sağlanmak istenmiştir. Döngüde `output_b(i|0b00000000);` komutu ile i değişkeni değeri 0b00000000 sayısı ile OR işlemine tabi tutulmuştur. B port'una gönderilen bilginin yüksek değerlikli ilk 3 bit'i (RB0, RB1, RB2) çoğullanan çıkış uçlarının seçimini ifade eder. B port'una gönderilen bilginin düşük değerlikli bilginin ilk üç bit'i (RB4, RB5, RB6) ise çoğullama için kullanılan 4 adet entegreyi seçmek için gerekli bilgiyi ifade eder. Örneğin ilk anda i değeri 0'dır ve bu değer 0b00000000 değeri ile OR'lanınca elde edilen değer 0b00000000 olur. Dolayısıyla elde edilen bilginin entegre seçme entegresine gelen düşük değerlikli ilk üç bit'ine bakıldığında birinci entegrenin seçildiğini, yüksek değerlikli ilk 3 bit'lerine bakıldığında seçilen entegrenin 0. çıkışının seçildiği görülüyor. Döngü ikinci dönüşünde i değeri 1 olacaktır. OR işlemi sonucunda ise bu sefer 0b00000001 bilgisi elde edilecektir. Buna göre de yine birinci entegre seçilmiş olacak ama bu sefer seçilen entegrenin 1. çıkışının seçildiği görülmektedir. Seçilen çıkış lojik-0 olacaktır ve çıkışa bağlı LED yanacaktır. Bu şekilde seçilen birinci 74138 entegresinin tüm 8 adet çıkışı sırayla lojik-0 olacak ve ilk döngüden çıkılacaktır.

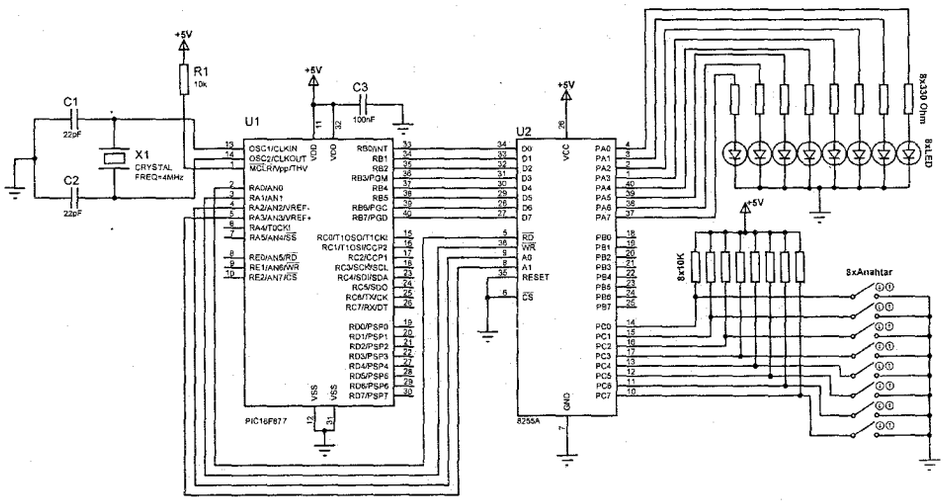
Programda ilk for döngüsünden çıkılında ikinci for döngüsüne gelinilecektir. İkinci for döngüsünde `output_b(i|0b00010000);` komutu ile bu sefer i değişkeni değeri 0b00010000 değeri ile OR'lanmaktadır. Bunun nedeni bu sefer de entegre seçici entegrenin ikinci 74138 entegresini aktif etmesi yani seçmesini sağlamaktır. Aynı mantıkla bu sefer ikinci 74138 entegresinin tüm çıkışları sırayla aktif yapılır. Tüm program bu mantık çerçevesinde çalışır ve tüm çıkışlar sırayla aktif olur.

8.6. 8255 İLE PORT ÇOĞULLAMA UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

8255 entegresinin diğer örneklerde incelenen çoğullama amaçlı entegrelerden daha karışık yapıya sahip olmasına rağmen daha güçlü bir kontrol imkânı sunan bir entegre olduğunu belirtmiştik. Bu uygulama da 8255 ile giriş/çıkış kontrol uygulaması yapılmıştır. Böylece 8255 entegresinin çalışması daha iyi anlaşılabilmektedir. Uygulamada 8255'in C port'u giriş, A port'u ise çıkış olarak yönlendirilmiştir. C port'una anahtarlar bağlanmıştır. Normalde anahtarlar açıkken A

port'una bağlı tüm LED'ler yanıktır. Hangi anahtar kapatılırsa o numaralı LED sönecektir.



Şekil-8.10. *8255 ile port çoğullama uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı       : 8255 İle Port Kontrol Uygulaması
*****************************************************************/
#include <16f877.h> // Kullanılacak denetleyicinin başlık
                    // dosyası tanıtılıyor.
//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              // osilatör frekansı belirtiliyor
#use standard_io (a) // Bu komut kullanıldığında A port'u için
            //giriş/çıkış yönlendirmesi      yapılmasına gerek yok
#use standard_io (b) // Bu komut kullanıldığında B port'u için
            //giriş/çıkış yönlendirmesi yapılmasına gerek yok

int data; // Tamsayı tipinde değişken tanımlanıyor

//************** 8255 OKUMA FONKSİYONU **************
void _8255_yazma (char x, char bilgi)
{
   switch (x)
   {
      case   'k':                    // Kaydediciye yazma seçilirse
             output_a(0b00001101);   // Kaydediciye yazma
             output_b(bilgi);        // Yazılacak bilgi gönderiliyor
             break;
```

```
        case    'a':                    // A port'una yazma seçilirse
                output_a(0b00000001);   // A port'una yazma
                output_b(bilgi);        // Yazılacak bilgi gönderiliyor
                break;

        case    'b':                    // B port'una yazma seçilirse
                output_a(0b00000101);   // B port'una yazma
                output_b(bilgi);        // Yazılacak bilgi gönderiliyor
                break;

        case    'c':                    // C port'una yazma seçilirse
                output_a(0b00001001);   // C port'una yazma
                output_b(bilgi);        // Yazılacak bilgi gönderiliyor
                break;

        default:
                break;
      }
}

//************** 8255 YAZMA FONKSİYONU ***************
int _8255_okuma (char x)
{
   switch (x)
   {
        case    'k':
                output_a(0b00001110); // Kaydediciden okuma
                return(input_b());    // Kaydedici değeri okunuyor
                break;

        case    'a':
                output_a(0b00000010);   // A port'undan okuma
                return(input_b());  // A port'u değeri okunuyor ve
                                    //fonksiyondan bu değerle çıkılıyor
                break;

        case    'b':
                output_a(0b00000110);   // B port'u okuma
                return(input_b());  // B port'u değeri okunuyor ve
                                    //fonksiyondan bu değerle çıkılıyor
                break;

        case    'c':
                output_a(0b00001010);   // C port'u okuma
                return(input_b());  // C port'u değeri okunuyor ve
                                    //fonksiyondan bu değerle çıkılıyor
                break;
        default:
                break;
   }
}
```

```
/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   _8255_yazma('k',0b10001001); // A port'u çıkış, C port'u giriş, B
                                //port'u çıkış yapılıyor
   _8255_yazma('a',0x00);       // A port'u çıkışı sıfırlanıyor

   while(1) // Sonsuz döngü
   {
      data=(_8255_okuma('c'));   // C port'u okunuyor
      _8255_yazma('a',data);    //A port'una okunan C port'u değeri
                                     //yazılıyor
   }
}
//*****************************************************************
```

Programın genel tanıtım ve ayar kısmında diğer programlarda kullanılan `#use fast_io(port ismi)` komutu bu programda kullanılmamıştır. Bu komut kullanıldığında programda hangi port'un giriş, hangi port'un çıkış olarak kullanılacağının belirtilmesi gerekliydi. Eğer programda `#use xxx_io()` komutlarından herhangi biri kullanılmamışsa programda port giriş/çıkış yönlendirilmesinin belirtilmesine gerek yoktu. Bu programda `#use standart_io()` komutu kullanılmıştır. Bunun anlamı, bu komutla belirtilen port'un yönlendirilmesinin derleyici program tarafından otomatik olarak yapılacağını belirtir. Örneğin programda `input_a()` komutu kullanılmışsa derleyici A port'unu giriş olarak yönlendirir, `output_a()` komutu kullanılmışsa derleyici A port'unu çıkış olarak yönlendirir. Programda hiçbir `#use xxx_io()` komutu kullanılmamışsa zaten derleyici otomatik olarak `#use standart_io()` komutunun kullanıldığını varsayılan ayar olarak kabul etmektedir. Programda bu komut bilgi olması açısından kullanılmıştır.

Programda 8255 entegresine istenen port'a veya kontrol kaydedicisine yazma ve okuma işlemleri için iki adet fonksiyon tanımlanmıştır. 8255 yazma işlemi için `_8255_yazma (char x, char bilgi)` fonksiyonu, yazma işlemi için `int _8255_okuma (char x)` fonksiyonu tanımlanmıştır. 8255 Okuma ve yazma fonksiyonlarında switch-case döngü yapısı kullanılmıştır. Yazma fonksiyonunun geri dönüş değeri olmadığından geri dönüş değeri `void` olarak belirlenmiştir. Fonksiyonun 2 adet parametresi yani değişkeni vardır. İki değişken de char tipinde tanımlanmıştır. Yazma fonksiyonu çağrıldığında ilk değişkeninin aldığı karakter

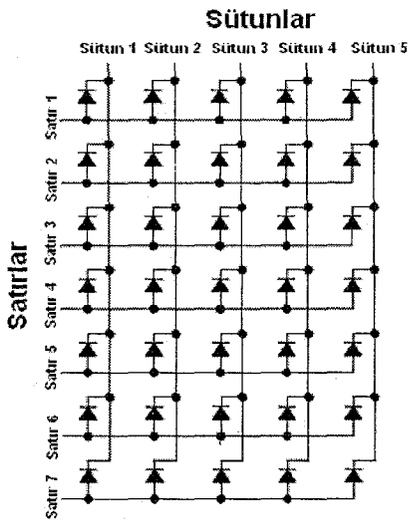
değerine göre switch-case döngüsü dallanma yapacaktır. Örneğin fonksiyon çağrılırken _8255_yazma(k,0x23) yazılmışsa, x değeri "k" olacak ve fonksiyon case 'k': kısmındaki komutları işleyecek ve break; komutu ile döngüden çıkacaktır. Diğer case durumlarını işlemeyecektir. Devre şemasında dikkat edilirse kontrol uçları ($\overline{RD}, \overline{WR}, A0, A1$) PIC16F877'in A port'una bağlanmıştır. Önceden verilen Tablo-8.4 incelenirse yazma işlemi için $\overline{RD}$ ucu (pin_A0) lojik-1, $\overline{WR}$ (pin_A1) bit'i lojik-0 olmalıdır. A0 ve A1 uçlarının alacağı değerler ise kaydediciye yazma, A, B ve C port'larına yazma için değişik değerlerde olmalıdır. Bu değerler istenen duruma göre fonksiyonda `output_a(0bxxxxxxxx);` komutları ile A port'una gönderilmektedir. Ardından `output_b(bilgi);`komutları ile de fonksiyon çağrıldığı sırada yazılan ikinci parametre değeri B port'una yani 8255'in D0-D7 uçlarına gönderilmektedir. Böylece 8255 yazma fonksiyonu oluşturulmuş olmaktadır.

8255 okuma fonksiyonu ise geri dönüş değeri int yani tamsayı olan ve tek parametreye yani değişkene sahip olan bir fonksiyon olarak tanımlanmıştır. Okuma fonksiyonunda da switch-case yapısı kullanılmıştır. Okuma fonksiyonunda da yazma fonksiyonunda olduğu gibi ilk başta kontrol bit'leri okuma durumuna göre programlanmıştır. Ardından `return(input_b());` komutları ile okunan değer ile fonksiyonun geri dönmesi sağlanmıştır. Örneğin programda `_8255_okuma('b')` şeklinde fonksiyon çağrıldığında, okuma fonksiyonu `case 'b':` kısmına gidecek ve kontrol uçlarını B port'unu okuma şeklinde ayarlayacaktır. Ardından da B port'unu okuyarak okuduğu değer ile geri dönecektir.

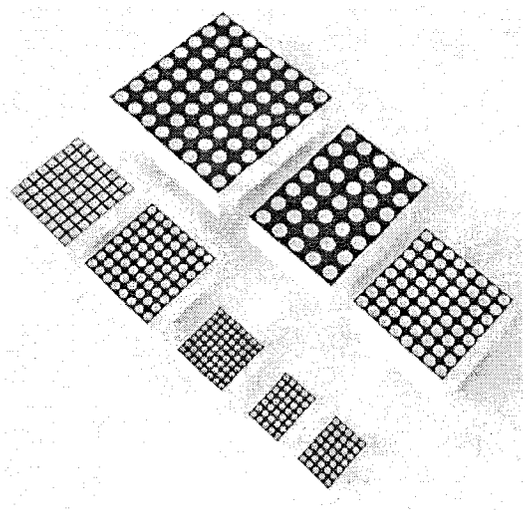
Ana fonksiyonda `_8255_yazma('k',0b10001001);` komutu ile Mod 0'da A ve B port'larını çıkış, C port'unu ise giriş olarak ayarlamak için gerekli kontrol kelimesi kontrol kaydedicisine yazdırılmıştır. Sonra oluşturulan sonsuz döngüde `data=(_8255_okuma('c'));` komutu ile 8255'in C port'u girişleri okunmuş ve okunan değer "data" adlı değişkene aktarılmıştır. `_8255_yazma('a',data);` komutu ile de "data" değişkeni değeri 8255'in A port'una yazılmış yani gönderilmiştir.

8.7. MATRİS LED DİSPLAY

Görüntüleme cihazları olarak kullanılan LED, 7 segment display ve LCD ekranlar her amaç için yeterli olmayabilir. Genellikle reklâm sektöründe görüntüleme amaçlı olarak çeşitli özellikte LED'ler matris yapısında birbirine bağlanır. Elde edilen bu LED bağlantısında istenen harf veya görüntüler elde edilir. Matris LED display'ler de, satır ve sütunları LED'lerden oluşan bir matris bağlantısını içinde barındıran tümleşik bir matris bağlantılı LED grubudur. Yani içinde matris yapısında bir bağlantıya sahip LED'ler içerirler. Piyasada 5x7 (5 sütun, 7 satır), 8x8 (8 sütun, 8 satır) gibi çeşitli yapıda matris LED display'ler vardır. Şekil-8.11'de 7x5 matris LED'in iç bağlantısı, Şekil-8.12'de ise çeşitli matris LED display'ler görülmektedir.

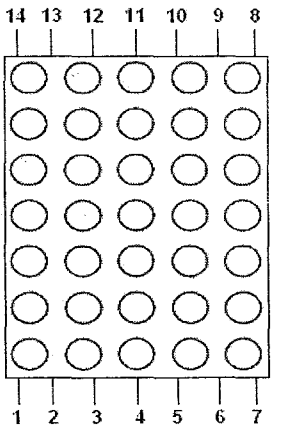


Şekil-8.11. *Matris display iç yapısı.*



Şekil-8.12. *Çeşitli matris display'ler.*

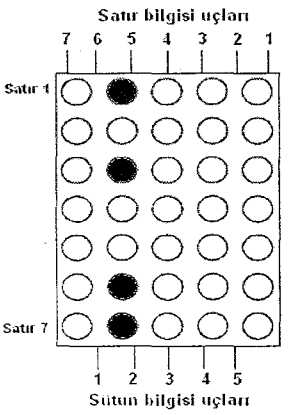
Matris LED display'ler sütun ve satır olmak üzere bilgi uçlarına sahiptir. Matris LED display'leri sürmek için tarama metodu kullanılır. Yani ilk başta ilk sütun aktif edilir daha sonra ilk sütunda bulunan istenen LED'i yakmak için ilgili satır veya satırlar aktif edilir. Ardından ikinci sütun aktif edilir ve bu sütunda yanması istenen LED'ler için gerekli satırlar aktif edilir. Bu işlemler tarama mantığına göre bu şekilde diğer sütunlar için de devam ettirilir. Böylece matris LED'de gösterilmek istenen karakter veya şekil oluşturulmuş olur. Bu şekilde yapılan tarama sonucunda elde edilen görüntünün göz tarafından algılanabilmesi için yeterli bir süre bu tarama işleminin tekrar edilmesi gerekmektedir. Matris LED display'leri sürmek için çeşitli entegrelerde mevcuttur. Örneğin MAX6952 ve MAX6953, maxim-dallas firmasının matris LED display'ler için sürücü entegre ürünleridir. **Katot sütun ve Anot sütun olmak üzere iki çeşit matris LED display vardır.** Katot sütun matris LED display'lerde sütunları aktif etmek için ilgili sütuna lojik-0, satırları aktif etmek için ilgili satıra lojik-1 uygulanmalıdır. Anot sütun matris LED display'lerde ise sütunları aktif etmek için ilgili sütuna lojik-1, satırları aktif etmek için ilgili sütuna lojik-0 uygulanmalıdır. Şekil-8.13'de 7x5 matris LED display'in uç numaraları Tablo-8.5'de ise uçların açıklamaları görülmektedir. Verilen şekil hem katot matris hem de anot matris için geçerlidir. Kullandığınız matris LED display'in durumuna göre istenen LED'i yakmak için sütun ve satır uçlarına uygun değer vermelisiniz (lojik-0 veya lojik-1). Dikkat edilirse normalde 7x5 matriste toplam 12 adet uç bulunması gerekir. Fakat 3. sütun ve 4. satır uçlarından ikişer uç vardır.



Pin No	Fonksiyon
1	5. Satır
2	7. Satır
3	2. Sütun
4	3. Sütun
5	4. Satır
6	5. Sütun
7	6. Satır
8	3. Satır
9	1. Satır
10	4. Sütun
11	3. Sütun
12	4. Satır
13	1. Sütun
14	2. Satır

Şekil-8.13. *7x5 Matris LED uç numaraları* ***Tablo-8.5.*** *7x5 Matris uç fonksiyonları*

Örneğin anot sütun bir matris LED display'de Şekil-8.14'teki gibi 2.sütunda gösterilen LED'leri yakmak için matris'in 2.sütun ucuna lojik-1 (+5V) sütun2=1, satır bilgisi olarak da istenen satırdaki LED'i yakmak için o satıra lojik-0 (şase) bilgisi gönderilmelidir. Bu mantıkla satır bilgisi olarak 0b0011010 (0x1A) değeri gönderilmelidir.

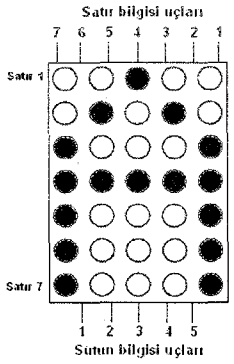


Şekil-8.14. *Matris LED display'de herhangi bir görünüm.*

Yine anot sütun bir matris LED display'de Şekil-8.15'deki "A" görünümü elde etmek için, tarama yöntemi ile sırayla aralarında belli bir gecikme vererek aşağıdaki bilgiler satır ve sütun hatlarına gönderilir. Matris LED display'de oluşturulan görüntünün yeterli bir süre göze görünmesi için aşağıda verilen döngü, komple belli bir sayıda örneğin 15 kez (mikrodenetleyicide kullanılan osilatör hızına bağlı olarak) tekrar edilmelidir. Eğer siz anot sütun değil de katot sütun bir matris LED display kullanıyorsanız. Aşağıda verilen bilgilerin tam tersini kullanmanız gerektiğini

unutmayın. Yani sütunları aktif etmek için lojik-1 yerine lojik-0, satırları aktif etmek için lojik-0 yerine lojik-1 vermeniz gerekir.

1.sütun aktif (sütun1=1) diğer sütunlar pasif (sütun2=0, sütun3=0, sütun4=0, sütun5=0), satır bilgisi olarak da 0b0000011 (0x03)
Gecikme
2.sütun aktif (sütun2=1) diğer sütunlar pasif (sütun2=0, sütun3=0, sütun4=0, sütun5=0), satır bilgisi olarak da 0b0000011 (0x75)
Gecikme
3.sütun aktif (sütun3=1) diğer sütunlar pasif (sütun2=0, sütun3=0, sütun4=0, sütun5=0), satır bilgisi olarak da 0b1110110 (0x76)
Gecikme
4.sütun aktif (sütun4=1) diğer sütunlar pasif (sütun2=0, sütun3=0, sütun4=0, sütun5=0), satır bilgisi olarak da 0b0000011 (0x75)
Gecikme
5.sütun aktif (sütun4=1) diğer sütunlar pasif (sütun2=0, sütun3=0, sütun4=0, sütun5=0), satır bilgisi olarak da 0b0000011 (0x03)
Gecikme

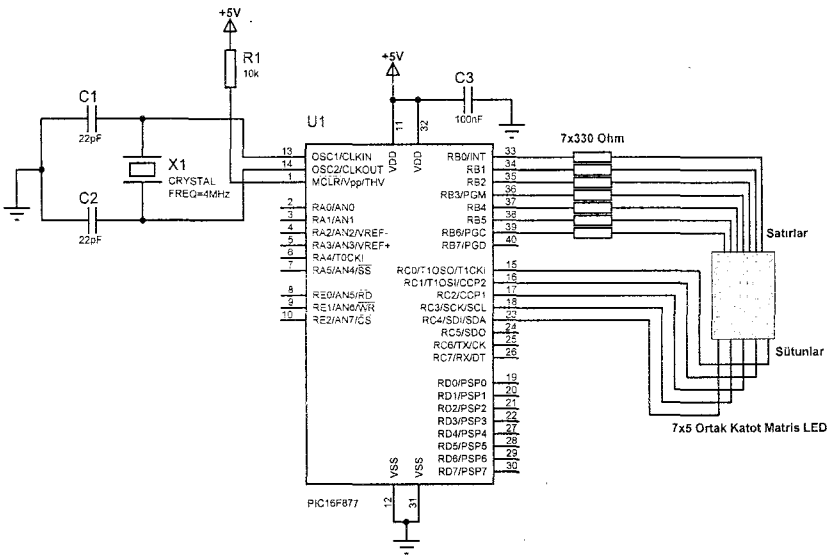


Şekil-8.15. *Matris LED display'de A karakteri görünümü.*

8.8. MATRİS LED DİSPLAY UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada 7x5 anot sütun bir matris LED display'in sürülmesi işlemi incelenmiştir. Uygulamada matris LED'de A, b, c, d, E, +, * ve çift yönlü ok karakterleri gösterilmiştir. Örnekte yukarıda anlatılan tarama metodu kullanılmıştır. Devre programında satır ve sütun veri değerlerini matris LED kısmında anlatılan mantığa göre kendi matris LED display'iniz için uygun hale getirebilirsiniz.



Şekil-8.16. *Matris LED display uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 Uygulama Adı       : Matris LED Display Uygulamas
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.

//***********Denetleyici konfigürasyon ayarları************
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              //osilatör frekansı belirtiliyor

#use fast_io(b) // Port yönlendirme komutları B port'u için geçerli
#use fast_io(c) // Port yönlendirme komutları C port'u için geçerli

//******** Matrise Gönderilecek Karakter Bilgileri *******
 int A[]={0x03,0x75,0x76,0x75,0X03};
 int B[]={0x00,0X36,0X36,0x36,0X49};
 int c[]={0x41,0x3E,0x3E,0x3E,0x5D};
 int D[]={0x00,0x3E,0x3E,0x3E,0x41};
 int E[]={0x00,0xB6,0xB6,0xB6,0xBE};
 int arti[]={0x77,0x77,0x41,0x77,0x77};
 int cift_ok[]={0x6B,0x5D,0x00,0x5D,0x6B};

//******************** Sütun Seçme Fonksiyonu *******************
int sutunsec (char s)
```

```
{
   switch (s)
   {
      case 0: return(0x10);break;  //1.Sütun aktif, diğerleri pasif
      case 1: return(0x08);break;  //2.Sütun aktif, diğerleri pasif
      case 2: return(0x04);break;  //3.Sütun aktif, diğerleri pasif
      case 3: return(0x02);break;  //4.Sütun aktif, diğerleri pasif
      case 4: return(0x01);break;  //5.Sütun aktif, diğerleri pasif
   }
}

//******* Matrise Veri Gönderme Fonksiyonu *******
void matris_karakter_gonder(char k[], int tekrar)
{
   int i,j;  // Değişkenler tanımlanıyor

   for (i=0;i<tekrar;i++) // Karakterin Matris LED'de görünmesi için
                          // tekrar döngüsü
       {
         for(j=0;j<5;j++) // Bu döngüde tarama işlemi yapılıyor
         {
            output_c(sutunsec(j)); // Sütun seç ve C port'una sütun
                                   // değerini gönder
            output_b(k[j]); // Gösterilecek harfin satır
                            // bilgisini B port'una gönder
            delay_ms(10);       // Gecikme veriliyor
         }
       }
}

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00); //B port'u tümüyle çıkış olarak yönlendiriliyor
   set_tris_c(0x00); //C port'u tümüyle çıkış olarak yönlendiriliyor

   while(1)      Sonsuz döngü oluşturuluyor
   {
     matris_karakter_gonder(A,25);// A karakterini matriste göster
     matris_karakter_gonder(B,25);// B karakterini matriste göster
     matris_karakter_gonder(C,25);// C karakterini matriste göster
     matris_karakter_gonder(D,25);// D karakterini matriste göster
     matris_karakter_gonder(E,25);// E karakterini matriste göster
```

```
        matris_karakter_gonder(arti,25);//+ karakterini matriste göster
        matris_karakter_gonder(cift_ok,25);// Çift yönlü ok karakterini
                                           // matriste göster
    }
}
//*****************************************************************
```

Devre programında ilk başta matris LED display'de gösterilecek karakterlerin bilgileri dizi şeklinde tanımlanmıştır. Karakter dizilerinde ki veri değerleri önceden anlatılan mantıkla hesaplanmıştır. Programda 2 adet fonksiyon tanımlanmıştır. Bunlardan birisi sütun seçme fonksiyonu (sutunsec()), diğeri matrise karakter gönderme fonksiyonudur (matris_karakter_gonderme()). Sütun seçme fonksiyonunda tek bir fonksiyon parametresi vardır. Bu da "s" değişkenidir. Fonksiyonda switch-case döngüsü kullanılmıştır. Fonksiyon çağrıldığında gelen parametre değerine göre ilgili sütun aktif edilmekte, diğer sütunlar pasif edilmektedir. Matrise veri yazma fonksiyonunda ise 2 adet parametre kullanılmıştır. Bu parametrelerden ilki gösterilecek karakter dizisinin seçimi için, diğer parametre de matris LED display'de görünecek karakterin görünme süresini belirtmesi içindir. Fonksiyonda `for (i=0;i<tekrar;i++)` komutu ile tekrar değişkeni değeri kadar tarama işleminin devam etmesi sağlanmıştır. Böylece her karakterin gösterimde kalma süresi bu döngü ile sağlanmıştır. İkinci `for(j=0;j<5;j++)` döngüsü ile de tarama işlemi yapılmıştır. Her döngü başında ilk başta sutunsec() fonksiyonu çağrılarak sütun seçilmekte ve ardından `output_b(harf[j]);` komutu ile de ilgili karakter bilgisi B port'una gönderilmektedir. Döngüde i değeri 0'dan başlayarak her döngü başına gelince 1 arttırılmaktadır. Böylece sırayla 5 sütun seçilmekte ve her sütunda yanması gereken satır LED'leri bilgisi ilgili sütun aktif iken B port'una gönderilmektedir. Her sütunun aktif kalma süresi 10msn olarak belirlenmiştir. Kendi sisteminizde bu süreyi değiştirebilirsiniz. Ana fonksiyonda ise sırayla matrise veri gönderme fonksiyonu `matris_karakter_gonder(A,25);` şeklinde çağrılmaktadır. Sonsuz döngü ile (while(1) ile verilen) program bu şekilde çalışmaya devam eder.

9. Bölüm

Kesme (INTERRUPT) İşlemleri

- Kesme Mantığına ve Kesme Olaylarına Genel Bakış
- CCS C'de Kesme Oluşturma İşlemleri
- enable_interrupts( ) Fonksiyonu
- disable_interrupts( ) Fonksiyonu
- clear_interrupts( ) Fonksiyonu
- #INT_XXX
- #PRIORITY
- #INT_EXT
- ext_int_edge( ) Fonksiyonu
- #INT_RB
- Dış Kesme (EXT_INT) Uygulaması
- RB4-RB7 Pin'lerindeki Değişim Kesmesi Uygulaması

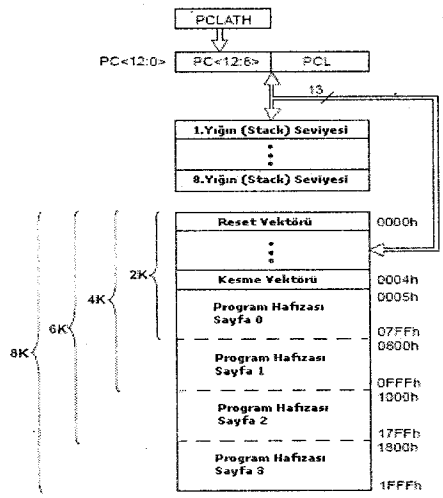
Bu bölümde kesme işlemleri ile ilgili temel bilgiler ile CCS C derleyicisinde kesme işlemlerinin nasıl yapıldığı anlatılmıştır. Ayrıca yine bu bölümde dış kesme uygulaması ve RB4...RB7 değişim kesmesi uygulaması ile iki adet kesme örneği uygulaması incelenmiştir. Bölümde anlatılan teorik bilgiler ve pratik uygulamalar ile tüm kesme işlemleri anlaşılabilir. Bu bölümde sadece 2 kesme uygulaması incelenmiştir. Diğer kesme çeşitleri ile aynı mantıkla kesme işlemleri yapılabilir. İleriki bölümlerde diğer tüm kesmeler ile ilgili örnekler ilgili bölümlerde incelenmiştir (Örneğin EEPROM yazma işlemi bitti kesmesi Dâhili EEPROM uygulaması örneğinde incelenmiştir).

9.1. KESME MANTIĞI VE KESME OLAYLARINA GENEL BİR BAKIŞ

Mikrodenetleyicinin değişik kaynaklardan gelen sinyaller ile mevcut programının çalışmasını kesip, önceden tanımlanmış kesme programını (fonksiyonunu) icra edip,

tekrar ana programa kaldığı yerden çalışmaya devam etmesi işlemini **kesme** olarak tanımlayabiliriz. Mikroişlemcili ve mikrodenetleyicili sistemlerde kesmeler olmazsa olmaz niteliktedir. Peki, neden kesmeler bu kadar önemlidir. Tasarlanan sistemlerde yapılan rutin kontrol işlemlerinin yanında çoğunlukla bazı giriş/çıkış uçları da denetlenmektedir -taranmaktadır-. Bu denetleme işlemleri sadece program vasıtası ile yapıldığında programın çalışma süresinin ve hızının neredeyse büyük bir kısmını işgal etmektedir. Bu kontrol veya tarama işlemi programda icra edilmesi gereken diğer işlerin yavaş yapılmasına veya kontrol edilmesi gereken diğer giriş/çıkış uçlarında veri kaybına yol açılmasına sebep olabilir. Örneğin bir sistemde, normal rutin programını icra eden bir programın aynı anda sistemde bulunan klavyeden basılan tuş değerini taradığını ve de başka port girişlerine bağlı buton girişlerini de taradığını düşünelim. Bu işleri tümüyle program vasıtası ile yaptığımızda klavye girişleri taranırken, buton girişleri taranamamaktadır veya her iki tarama işlemi yapılırken program diğer komutlarını icra edememektedir. Yani program ile sadece bir iş kontrol edilebilmektedir. Program vasıtası ile yapılan, belirli zaman aralıklarında sürekli olarak kontrol edilen bu işleme **yoklama** denilir. Yoklama işlemi ile sadece tek bir işlem icra edilebildiğinden, mikrodenetleyiciler de çeşitli kesme donanım birimleri bulunmaktadır. Program ile yapılması gereken işlemleri denetleyici içinde bulunan kesme donanım birimleri yaparsa, sistemde aynı anda birden fazla işlem yapılabilir. Örneğin bir buton girişini denetleyici içinde bulunan bir donanım birimi kontrol ederse, program bu iş için meşgul edilmeyecek ve başka işlemler için kullanılacaktır. Yukarıda anlatılan nedenlerden dolayı **yoklama** işlemi yerine **kesme** kullanmak çok daha verimli ve esnek programlar yazılmasını sağlar.

PIC denetleyicilere ilk enerji geldiğinde veya PIC sıfırlandığında (resetlendiğinde) program sayıcı (Program Counter-PC, çalıştırılan komutun adresini tutan kaydedici) 0000h adresini gösterir. Bu adrese **reset vektörü** denir. Bu 0000h adresine PIC'e yüklenen programın başlangıç adresi yüklenir. Böylece denetleyici ilk enerjilendiği anda ilk başta 0000h adresine bakar ve bu adresin içinde bulunan adres değerine giderek programını icra eder. Yukarıda bahsedilen kesme durumlarından biri oluştuğunda da program sayıcı (PC) program hafızasında 0004h adresine gider. Bu adrese **kesme vektörü** denir. Bu adreste de kesme alt programının (fonksiyonunun) başlangıç adresi vardır. Bu nedenle program yazarken bir kesme olayını programımızda aktif etmişsek, 0004h adresine mutlaka kesme alt programımızın başlangıç adresini yazmamız gerekir. Bu sayede kesme durumu meydana geldiğinde PC, 0004h adresini gösterecek, program 0004h adresine gidecek ve bu adresteki kesme programı adresine dallanacak ve kesme programını icra edecektir. Anlatılan 0000h adresindeki reset vektörü ve 0004h adresindeki kesme vektörü tüm PIC denetleyicilerinde aynı adrestedir. Şekil-9.1'de PIC16F877'in program hafızasında anlatılan reset vektörü ve kesme vektörü görülmektedir.



Şekil-9.1. *PIC16F877 program hafızası.*

Herhangi bir kesme sinyali geldiğinde ana program çalışması kesiliyor, kesme alt programına gidiliyor, kesme alt programı icra ediliyor ve ana programa geri dönülüyor. **Peki denetleyici kesme sinyali geldiğinde icra etmeyi bıraktığı ana programda kaldığı yeri nasıl biliyor?** Kesme meydana geldiği anda denetleyici içindeki program sayıcı (PC) o anki değerini **yığın (stack)** kaydedicisine atar. Kesme programı dönüşünde yığından kaydedilen adres alınır ve bu adresteki komuttan itibaren ana program icra edilmeye devam eder. Şekil-8.1'de de görüldüğü gibi yığın kaydedicisi 8 seviyeden oluşmaktadır. Buradan da anlaşıldığı gibi yığına üst üste sadece 8 adet adres bilgisi kaydedilebilir.

Denetleyicide bir kesme meydana geldiğinde o kesmeye ait bayrak (flag) "1" olur. Anlamı kolaylaştırmak açısından kesme meydana geldiğinde kesmenin bayrağı kalkar diyebiliriz. Biz hangi kesme bayrağı kalkmış yani "1" olmuşsa o kesmenin oluştuğunu anlarız. PIC denetleyicilerde 14 adet kesme mevcuttur. Bu kesmeler şunlardır.

1. RB0 Harici Kesmesi.
2. RB4-RB7 Pin'lerindeki Değişiklik Kesmesi.
3. Timer0 Birimi Taşma Kesmesi.
4. Timer1 Birimi Taşma Kesmesi.
5. Timer2 Birimi Taşma Kesmesi.
6. A/D Çevrimi Yapıldığında Meydana Gelen Kesme.
7. CCP1 Modülü Kesmesi.
8. CCP2 Modülü Kesmesi.
9. Paralel Port'tan Veri Gelme Kesmesi.
10. Seri Port'tan Veri Geldiğinde Oluşan Kesme.
11. SPI veya I^2C İletişimi Sırasında Veri Gelme Kesmesi.
12. EEPROM'a Veri Yazma İşlemi Sonlandığında Oluşan Kesme.

13. RS232 Seri İletişiminde Gönderilecek Veri Tamponunu Boş Olduğunda Meydana Gelen Kesme.
14. Bus Collision Kesmesi (MSSP modunda seri iletişimde hata oluştuğunda).

Bu kesmelerden ilk 3 kesme olan RB0 harici kesmesi, RB4-RB7 değişim kesmesi ve Timer0 taşma kesmeleri **INTCON** ve **OPTION** kaydedicileri ile kontrol edilir. Diğer 11 kesme ise **çevresel kesme (Peripheral Interrupt) kaynaklarını** oluşturur. Çevresel kesme kaynaklarını kontrol eden kesme kaydedicileri ise **PIE1, PIR1, PIE2 ve PIR2** kaydedicileridir. Bu kaydedicilerin içerikleri aşağıda sırayla açıklanmıştır. Bu çevresel kesme kaynakları kaydedicileri ile, kesmelerin aktif ve pasif yapılması kontrolü sağlanır ve kesme sinyali geldiğinde kesmenin hangi kesme olduğunu bize bildirir. **PIE1 ve PIE2 kaydedicileri çevresel kesmeleri aktif etmek için, PIR1 ve PIR2 kaydedicileri ise kesme bayraklarını içerir. Yani oluşan kesmenin hangi kesme olduğunu bu bayrakları kontrol ederek anlarız.**

Aşağıda verilen kaydedici tanımlamaları tümüyle bilgi amaçlıdır. CCS C'de bu kaydedicilerin içeriklerinin anlamlarını bilmeye gerek yoktur. Daha basit bir şekilde kesme işlemleri yapılabilmektedir.

Şekillerde ki, **R=** Okunabilir bit, **W=** İlgili bit'e bilgi yazılabilir, **U=** Kullanılmayan bit, **-0=** İlk enerji geldiğinde "0" olarak okunur, **-x=** İlk enerji geldiğinde durumu belli değil, anlamındadır.

INTCON KAYDEDİCİSİ

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Şekil-9.2. *INTCON kaydedicisi.*

Bit 7 **GIE** = Tüm kesmelere izin verme bit'i.
0: Hiçbir kesmeye izin verilmez.
1: Aktif edilmiş tüm kesmelere izin verilir.

Bit 6 **PEIE** = Çevresel kesmelere izin verme bit'i.
0: Hiçbir çevresel kesmeye izin verilmez.
1:Aktif edilmiş tüm çevresel kesmelere izin verilir.

Bit 5 **TOIE** = TMR0 taşma kesmesi izin verme bit'i.
0:TMR0 taşma kesmesine izin verilmez.
1: TMR0 taşma kesmesine izin verilir.

Bit 4 **INTE** = RB0/INT dış kesmesi izin verme bit'i.
0:RB0/INT dış kesmesine izin verilmez.
1: RB0/INT dış kesmesine izin verilir.

Bit 3 **RBIE** = RB4-RB7 pin'lerindeki değişim kesmesine izin verme bit'i.
0: RB4-RB7 pin'lerindeki değişim kesmesine izin verilmez.
1: RB4-RB7 pin'lerindeki değişim kesmesine izin verilir.

Bit 2 **TOIF** = TMR0 taşma kesmesi bayrağı bit'i.

0:TMR0 zaman aşımı yok.
1:TMR0 zaman aşımı var.

Bit 1 **INTF** = RB0/INT dış kesmesi bayrağı bit'i.
0: RB0/INT dış kesmesi oluşmamış.
1: RB0/INT dış kesmesi oluşmuş.

Bit 0 **RBIF** = RB4-RB7 pin'lerindeki değişim kesmesi bayrağı bit'i.
0: RB4-RB7 pin'lerindeki değişim kesmesi oluşmamış.
1: RB4-RB7 pin'lerindeki değişim kesmesi oluşmuş.

OPTION KAYDEDİCİSİ

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Şekil-9.3. *OPTION kaydedicisi.*

Bit 7 **RBPU** = PORTB Pull-up yapma bit'i.
0: Pull-up açık.
1: Pull-up kapalı.

Bit 6 **INTEDG** = RB0/INT pin'indeki kesme sinyalinin aktif olduğu kenarı seçme bit'i.
0: Kesme düşen kenarda aktif.
1: Kesme yükselen kenarda aktif.

Bit 5 **TOCS** = TMR0 sinyal kaynağı seçme bit'i.
0: Dahili kaynak.
1: Harici sinyal kaynağı.

Bit 4 **TOSE** = TMR0 sinyal kaynağı kenar seçme bit'i
0: Düşen kenar tetiklemesi.
1: Yükselen kenar tetiklemesi

Bit 3 **PSA** = Frekans bölücü seçme bit'i.
0: Frekans bölme sayısı TMR0 için geçerli.
1: Frekans bölme sayısı WDT için geçerli.

Bit 2,1,0 **PS2, PS1, PS0** = Frekans bölme sayısı (Prescaler value)

Bit Değeri (PS2,PS1,PS0)	TMR0 ORANI	WDT ORANI
000	1:2	1.1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

PIE1 KAYDEDİCİSİ

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Şekil-9.4. *PIE1 kaydedicisi.*

Bit 7 **PSPIE** = Paralel Slave Port Okuma/Yazma kesmesi izin verme bit'i.
0:PSP okuma/yazma kesmesine izin verilmez.
1:PSP okuma/yazma kesmesine izin verilir.

Bit 6 **ADIE** = A/D çevrimi kesmesi izin verme bit'i.
0: A/D çevrimi kesmesine izin verilmez.
1: A/D çevrimi kesmesine izin verilir.

Bit 5 **RCIE** = USART bilgi alındı kesmesi izin verme bit'i.
0: USART bilgi alındı kesmesine izin verilmez.
1: USART bilgi alındı kesmesine izin verilir.

Bit 4 **TXIE** = USART bilgi gönderme kesmesi izin verme bit'i.
0: USART bilgi gönderme kesmesine izin verilmez.
1: USART bilgi gönderme kesmesine izin verilir.

Bit 3 **SSPIE** = Senkronize seri port kesmesi izin verme bit'i.
0: Senkronize seri port kesmesine izin verilmez.
1: Senkronize seri port kesmesine izin verilmez.

Bit 2 **CCP1IE** = CCP1 kesmesi izin verme bit'i.
0: CCP1 kesmesine izin verilmez.
1: CCP1 kesmesine izin verilir.

Bit 1 **TMR2IE** = TMR2 değerinin PR2 değerine eşit olduğunda kesme meydana gelmesi izin bit'i.
0:TMR2 değeri PR2 değerine eşit olunca kesme oluşmaz.
1:TMR2 değeri PR2 değerine eşit olunca kesme oluşur.

Bit 0 **TMR1IE** =TMR1 taşma kesmesi izin verme bit'i.
0: TMR1 taşma kesmesine izin verilmez.
1: TMR1 taşma kesmesine izin verilir.

PIR1 KAYDEDİCİSİ

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Şekil-9.5. *PIR1 kaydedicisi.*

Bit 7 **PSPIF** = Paralel Slave Port Okuma/Yazma kesmesi bayrağı bit'i.
0: PSP okuma/yazma işlemi meydana gelmemiş.
1: PSP okuma/yazma işlemi meydana gelmiş.

Bit 6 **ADIF** = A/D çevrimi kesmesi bayrağı bit'i.
0: A/D çevrimi tamamlanmamış.
1: A/D çevrimi tamamlanmış.

Bit 5 **RCIF** = USART bilgi alındı kesmesi bayrağı bit'i.

0: USART bilgi alma tamponu boş.
1: USART bilgi alma tamponu dolu.

Bit 4 **TXIF** = USART bilgi gönderme kesmesi bayrağı bit'i.
0: USART bilgi gönderme tamponu boş.
1: USART bilgi gönderme tamponu dolu.

Bit 3 **SSPIF** = Senkronize seri port (SSP) kesmesi bayrağı bit'i.
0: SSP kesmesi şartı oluşmamış.
1: SSP kesmesi şartı oluşmuş. Şartlar şunlardan biri olmalıdır;
SPI biriminde bilgi alma/gönderme olursa.
I^2C Slave konumunda bilgi alma/gönderme olursa.
I^2C Master konumunda iken,
- Bilgi alma/gönderme olursa
- START, STOP ve Yeniden kurma durumlarında

Bit 2 **CCP1IF** = CCP1 kesmesi bayrağı bit'i.
0: CCP1 içeriği, TMR1 içeriğini yakalamamıştır.
1: CCP1 içeriği, TMR1 içeriğini yakalamıştır.

Bit 1 **TMR2IF** = TMR2 değerinin PR2 değerine eşit olması kesmesi bayrağı.
0: TMR2 değeri PR2 değerine eşit değildir.
1: TMR2 değeri PR2 değerine eşittir.

Bit 0 **TMR1IF** = TMR1 taşma kesmesi bayrağı.
0:TMR1'de taşma oluşmamıştır.
1:TMR1'de taşma oluşmuştur.

PIE2 KAYDEDİCİSİ

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

Şekil-9.6. *PIE2 kaydedicisi.*

Bit 7 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 6 Ayrılmış bit.
Bit 5 Kullanılmayan bit. Okunduğunda "0" değeri alınır.

Bit 4 **EEIE** = EEPROM yazma işlemi bitti kesmesi izin verme bit'i.
0: EEPROM yazma işlemi bittiğinde kesme oluşmasın.
1: EEPROM yazma işlemi bittiğinde kesme oluşsun.

Bit 3 **BCLIE** = MSSP modunda Bus Collision hatası kesme izin verme bit'i.
0: Bus Collision hatası oluştuğunda kesme oluşmasın.
1: Bus Collision hatası oluştuğunda kesme oluşsun.

Bit 2 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 1 Kullanılmayan bit. Okunduğunda "0" değeri alınır.

Bit 0 **CCP2IE** = CCP2 modülü "capture" modunda iken CCPR2 kaydedicisindeki değer TMR2 kaydedicisindeki değere eşit olduğunda kesme oluşmasına izin verme bit'i.
0: CCP2 kesmesi oluşsun.
1: CCP2 kesmesi oluşmasın.

PIR2 KAYDEDİCİSİ

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

Şekil-9.7. *PIR2 kaydedicisi.*

Bit 7 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 6 Ayrılmış bit.
Bit 5 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 4 **EEIF** = EEPROM'a yazma işlemi bitti kesme bayrağı bit'i.
0: EEPROM yazma işlemi bitmemiş.
1: EEPROM yazma işlemi bitmiş.
Bit 3 **BCLIF** = MSSP modunda Bus Collision hatası kesme bayrağı bit'i.
0: Bus Collision hatası oluşmamış.
1: Bus Collision hatası oluşmuş.
Bit 2 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 1 Kullanılmayan bit. Okunduğunda "0" değeri alınır.
Bit 0 **CCP2IF** = CCP2 kesme bayrağı bit'i.
0: CCP2 kesmesi oluşmamış.
1: CCP2 kesmesi oluşmuş.

9.2. CCS C'DE KESME OLUŞTURMA İŞLEMLERİ

CCS C'de bir kesme komutu kullanıldığında ilgili kesme meydana geldiğinde işlenecek komutları içinde barındıran bir kesme fonksiyonunun tanımlanması gereklidir. CCS C'de kullanılacak kesme fonksiyonları mutlaka ana fonksiyondan önce tanımlanmalıdır. Kesme fonksiyonu tanımlanırken aşağıda verilen yapı kullanılır.

```
int_xxx          // xxx yerine ilgili kesme ismi yazılır.
[fonksiyon geri dönüş değeri türü] [fonksiyon ismi] ( )
{
        Komut veya komutlar;
}
```

Örnek bir kesme fonksiyonu;

```
#int_ext
void  dis_kesme ()
{
        output_low(pin_d1);
        x++;
}
```

#int_xxx komutunda aktif edilen fonksiyon ismi yazılır. Bu kesme isimleri kullanılan denetleyicinin tanıtım dosyasında (örneğin 16f877.h dosyasının içinde) tanımlanmıştır. PIC16F877 için aşağıda verilen kesme komutları kullanılır.

#INT_EEPROM → EEPROM yazma işlemi bitince meydana gelen kesme.
#INT_EXT → Dış kesme.
#INT_I2C →I2C Kesmesi.
#INT_LOWVOLT → Düşük voltaj tespit kesmesi.
#INT_RB → B Portunun B4-B7 pin'lerinde herhangi bir değişiklik olduğunda meydana gelen kesme.
#INT_RC → C Port'unun C4-C7 pin'lerinde herhangi bir değişiklik olduğunda meydana gelen kesme.
#INT_RDA → RS232 data alma kesmesi.
#INT_RTCC → Timer0 kesmesi.
#INT_SSP → SPI veya I2C faaliyette kesmesi.
#INT_TBE RS232 → gönderme tamponu boş kesmesi.
#INT_TIMER0 → Timer 0 kesmesi.
#INT_TIMER1 → Timer 1 kesmesi.
#INT_TIMER2 → Timer 2 kesmesi.
#INT_CCP1 → CCP1 kesmesi.
#INT_CCP2 → CCP2 kesmesi.

Kesmelerin aktif veya pasif yapılması için aşağıdaki komutlar kullanılır;

#enable_interrupt(*kesme ismi*); // Kesmeyi aktif yapmak için
#disable_interrupt(*kesme ismi*); // Kesmeyi pasif yapmak için.

```
#enable_interrupt(int_ext); // RB0/INT dış kesmesi aktif.
#disable_interrupt(int_ext); // RB0/INT dış kesmesi pasif.
```

PIC denetleyicilerde kesmeleri aktif etmek yetmez. INTCON kaydedicisinin 7.bit'i olan GIE bit'i aktif edilen tüm kesmelere izin vermek veya vermemek için kullanılır. CCS C'de aşağıdaki komutlarla tüm aktif edilmiş kesmelere izin verilir veya verilmez.

#enable_interrupt(GLOBAL); // Aktif edilen kesmelere izin verilir.
#disable_interrupt(GLOBAL); // Aktif edilen kesmelere izin verilmez.

Bir kesme meydana geldiğinde o kesme fonksiyonu icra edilirken başka bir kesme meydana gelse dahi aktif olamaz. İlk başta icra edilen kesme fonksiyonunun bitmesi gereklidir. Fakat aynı anda meydana gelen kesmeler de hangi kesmenin geçerli olacağını belirlemek için kesmelere öncelik verilmelidir. Öncelik vermek için aşağıdaki komut kullanılır. İlk başa yazılan kesme en fazla önceliğe sahiptir demektir.

#priority *kesme ismi, kesme ismi,...*

```
#priority ext, timer0   // dış kesme, timer0 kesmesinden daha
                        // öncelikli
```

PIC denetleyicilerde bir kesme meydana geldiğinde o kesmeye ait kesme bayrağı bit'i lojik-1 olur. Bunun nedeni kesme komutları işlenirken yine aynı kesme meydana gelirse kesme komutları icra edilirken kesmeden çıkılmamasıdır. Kesme komutlarının sonunda bu bayrak bit'inin lojik-0 yapılması gerekir. Eğer bu bit lojik-0 yapılmazsa, aynı kesme şartı oluşsa da dahi ikinci kez kesme meydana gelmez. CCS C derleyicisinde ise herhangi bir kesme meydana geldiğinde, CCS C programı kesme fonksiyonu çıkışında otomatik olarak kesme bayrağını siler. Kullanıcının bu işlemi komut ile yapmasına gerek kalmaz. Fakat yine de istendiğinde kesme bayrağını silmek için aşağıdaki komut kullanılır.

clear_interrupt(*kesme ismi*);

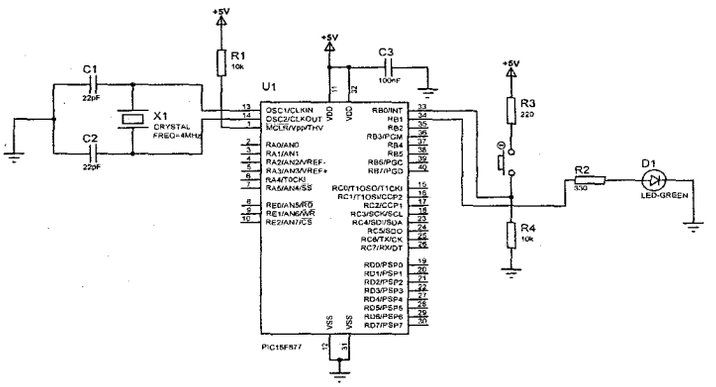
```
clear_interrupt(int_timer0); // Timer0 kesmesi bayrağı
                             // lojik-0 yapılır.
```

9.3. DIŞ KESME (EXTERNAL INTERRUPT-EXT_INT) UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

PIC ürünlerinin genelinde dış kesme pin'i bulunur. Bu pin RB0/INT pin'idir. Bu uygulamada RB0 ucuna bir buton bağlanmıştır. Bu butona basıldığında dış kesme (ext_int) oluşması sağlanmış ve RB1 pin'ine bağlı LED'in yanması sağlanmıştır. Programla kesme fonksiyonunda LED 1 saniye yakıldıktan sonra 3 saniye süre ile LED söndürülmüştür. Ardından LED 500 msn ara ile 10 kez yanıp söndürülmüştür. Böylece kesme fonksiyonu icra edilirken butona basılırsa yeni bir kesme oluşmayacağı görülebilir. Yeni bir dış kesmeye denetleyicinin cevap vermesi için programın kesme fonksiyonundan çıkmış olması gerekir.

Devreyi PIC PROG/DEKA 16F87x deneme kartında denemek için küçük bir işlem yapılması gereklidir. Bir zil teli kablosu ile deneme kartının sol kenarında bulunan RA0 çıkış pin'i ile deneme kartının sağ kenarında bulunan RB0 çıkış pin'i birbirine bağlanmalıdır. Ayrıca deneme kartındaki Port B jumper'ı LED konumunda olmalı.



Şekil-9.8. *Dış kesme uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
Uygulama Adı        :PIC16F877 ile Dış Kesme (INT_EXT) Uygulaması
PIC PROG/DEKA       : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         //dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           //osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

int i;  // Tamsayı tipinde değişken tanımlanıyor

//****** Dış Kesme Fonksiyonu ****************
#int_ext                 // Dış (External) RB0/INT kesmesi
void  ext_kesmesi ()     // Dış kesme fonksiyonu
{
   output_high(pin_b1);    // RB1 çıkışı "1" yapılıyor
   delay_ms(1000);         // 1 sn bekleme yapılıyor
   output_low(pin_b1);     // RB1 çıkışı "0" yapılıyor
   delay_ms(3000);         // 3 sn bekleme yapılıyor

   for (i=0;i<10;i++)
   {
      output_high(pin_b1);// RB1 çıkışı "1" yapılıyor
      delay_ms(500);      // 500 msn bekleme yapılıyor
      output_low(pin_b1); // RB1 çıkışı "0" yapılıyor
      delay_ms(500);      // 500 msn bekleme yapılıyor
   }
}
/********* ANA PROGRAM FONKSİYONU********/
```

```
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x01);    // RB0 pin'i giriş olarak yönlendiriliyor
   output_b(0x00);      // B port'u çıkışı ilk anda sıfırlanıyor

   ext_int_edge(H_TO_L);    // INT_EXT kesmesinin düşen kenarda aktif
                            // olacağını belirtir
   enable_interrupts(INT_EXT);  // INT_EXT kesmesini aktif yapar
   enable_interrupts(GLOBAL);   // Aktif edilen kesmelere izin ver

   while(1);    // Sonsuz döngü
}
//*******************************************************************
```

Programda ana fonksiyondan önce `#int_ext` komutu ile kesme oluştuğunda icra edilecek komutları barındıran kesme fonksiyonu tanımlanmıştır. Ana fonksiyonda ise `ext_int_edge(H_TO_L);` komutu ile dış kesmenin, giriş sinyalinin yüksek kenardan düşük kenara geçişi sonunda olması sağlanmıştır. Eğer dış kesmenin düşük kenardan yüksek kenara olmasını isteseydik `ext_int_edge(L_TO_H);` komutunu kullanmak gerekliydi. Bu işlemden sonra `enable_interrupts(INT_EXT);` komutu ile dış kesmeyi (RB0/INT) aktif hale getirmiş oluruz. Fakat istenen kesmeyi aktif hale getirmek yetmez. PIC denetleyicilerde birde tüm aktif edilmiş kesmelere izin vermek gerekir. Bu işlem için de program da `enable_interrupts(GLOBAL);` komutu kullanılmıştır. En son olarak da `while(1);` komutu ile sonsuz döngü oluşturulmuş ve programın bu şekilde devam etmesi sağlanmıştır.

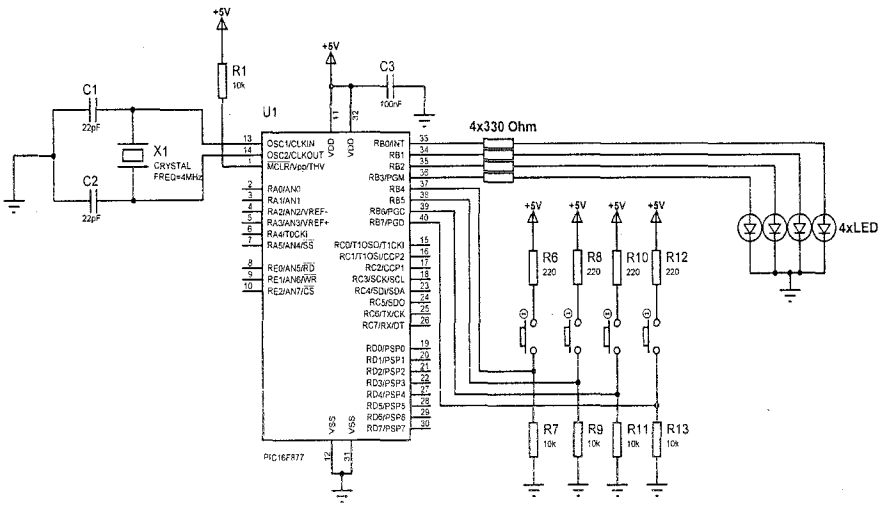
9.4. B PORT'UNDAKİ B4...B7 PİN'LERİNDE DEĞİŞİKLİK KESMESİ UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

PIC denetleyicilerde B port'unun B4, B5, B6 ve B7 pin'lerinden birinin durumu değiştiğinde (örneğin giriş değeri lojik-1 iken lojik-0 veya lojik-0 iken lojik-1 olursa) B port'u değişiklik kesmesi oluşturulabilir. Bu uygulamada bu kesme aktif edilmiş ve B4, B5, B6, B7 pin girişlerine bağlı butonlarla bu uçlarda değişiklik yapılması sağlanmıştır. B4...B7 uçlarından hangisinde değişiklik meydana gelirse yani hangi butona basılırsa o numaralı RB0, RB1, RB2 veya RB3 pin'lerine bağlı LED 2 saniye

süre ile yanmakta ve sonra söndürülmektedir. Böylece hangi pin'de değişiklik meydana geldiği LED'lerde görünmektedir.

Devreyi PIC PROG/DEKA 16f87x deneme kartında denemek için küçük bir işlem yapılması gereklidir. Zil teli kablosu ile deneme kartının sol kenarında bulunan RA0 çıkışını deneme kartının sağ tarafında bulunan RB4 ucuna, RA1 çıkışını RB5 ucuna, RA2 çıkışını RB6 ucuna, RA3 çıkışını RB7 ucuna bağlamanız gereklidir. Ayrıca deneme kartındaki Port B jumper'ı LED konumunda olmalı.



Şekil-9.9. *RB4...RB7 pin'lerindeki değişim kesmesi uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile B portu RB4...RB7 Değişiklik Kesmesi Uygulaması
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000)  // Gecikme fonksiyonu için kullanılacak
                            //osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

//*Port B RB4, RB5, RB6 ve RB7 pin'leri değişiklik Kesme Fonksiyonu
#int_RB // PORTB de B4..B7 pin'lerinde meydana gelen değişim kesmesi
void  B_degisiklik ()    // Kesme fonksiyonu ismi
{
   if (input(pin_b4))       // RB4 pin'i girişi okunuyor.
      output_high(pin_b0);  // RB0 çıkışı lojik-1 yapılıyor.
   if (input(pin_b5))       // RB5 pin'i girişi okunuyor.
```

```
      output_high(pin_b1); // RB1 çıkışı lojik-1 yapılıyor.
   if (input(pin_b6))      // RB6 pin'i girişi okunuyor.
      output_high(pin_b2); // RB2 çıkışı lojik-1 yapılıyor.
   if (input(pin_b7))      // RB7 pin'i girişi okunuyor.
      output_high(pin_b3); // RB3 çıkışı lojik-1 yapılıyor

   delay_ms(2000);    // 2 sn gecikme veriliyor.
   output_b(0x00);    // B port'u çıkışı komple sıfırlanıyor.
}
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0xF0);    // RB7,RB6,RB5,RB4 giriş, diğer uçlar çıkış

   output_b(0x00);      // B port'u çıkışı ilk anda sıfırlanıyor

   enable_interrupts(INT_RB);   // INT_EXT kesmesini aktif yapar
   enable_interrupts(GLOBAL);   // Aktif edilen kesmelere izin ver

   while(1);    // Sonsuz döngü
 }
//*****************************************************************
```

Devre de buton girişleri RB4...RB7 girişlerine bağlanmıştır ve girişler normalde lojik-0'dır. Butona basılınca giriş lojik-1 olur. Yani girişlerde değişiklik meydana gelmiş olur. Programda `#int_RB` ve `void B_degisiklik ()` ifadeleri ile RB4...RB7 değişiklik kesmesi fonksiyonu oluşturulmuştur. Kesme fonksiyonunda `if (input(pin_b4))` komutu ile RB4 girişi okunmakta ve giriş değeri "1" ise `output_high(pin_b0);` komutu işlem görmektedir. Bu komut ile RB0 çıkışı lojik-1 yapılmakta yani bu çıkışa bağlı LED yanmaktadır. Eğer RB4 giriş değeri "1" değilse `output_high(pin_b0);` komutu işlem görmeyecek diğer if komutuna atlayacaktır. Diğer if komutları ile de RB5, RB6 ve RB7 girişleri de okunmaktadır. Hangi giriş "1" ise ilgili LED yakılmaktadır. Böylece hangi pin'de değişiklik meydana geldiği RB0, RB1, RB2 ve RB3 pin'lerine bağlı LED'lerde görülebilmektedir. Meydana gelen değişikliğin rahatça görülebilmesi için kesme fonksiyonunda if komutlarından sonra `delay_ms(2000);` komutu ile 2 saniye gecikme verilmiştir. Yeni bir kesmenin oluştuğunun anlaşılması için kesme fonksiyonu çıkışında `output_b(0x00);` komutu ile B port'u çıkışları komple sıfırlanmıştır. Ana programda sadece `while(1);` komutu ile sonsuz döngü oluşturulmuş ve programın bu şekilde çalışmaya devam etmesi sağlanmıştır.

10. Bölüm

ZAMANLAYICI VE SAYICI İŞLEMLERİ

- Zamanlayıcı / Sayıcı Kavramları
- Timer0 Birimi ve Timer0 Uygulaması
- #INT_TIMER0 Fonksiyonu
- SETUP_TIMER_0()Fonksiyonu
- SET_TIMER0()Fonksiyon
- Timer1 Birimi ve Timer1 Uygulaması
- #INT_TIMER1 Fonksiyonu
- SETUP_TIMER_1() Fonksiyonu
- SET_TIMER1()Fonksiyonu
- Timer2 Birimi ve Timer2 Uygulaması
- #INT_TIMER2 Fonksiyonu
- SETUP_TIMER_2() Fonksiyonu
- SET_TIMER2()Fonksiyonu
- GET_TIMERx() Fonksiyonu
- Watchdog Timer (WDT) Birimi ve WDT Uygulaması
- SETUP_WDT() Fonksiyonu
- RESTART_WDT() Fonksiyonu
- SLEEP() Fonksiyonu

10.1. ZAMANLAYICI/SAYICI KAVRAMLARI

Zamanlayıcı (timer) ve sayıcı (counter) birimleri hem mikroişlemci hem de mikrodenetleyiciler için neredeyse bulunması zorunlu birimlerdir. Bu birimler için çok önceleri harici özel entegreler kullanılırdı. Mikrodenetleyici yapılarının gelişim evresinde önemli bir yer de bu tip donanım birimlerinin mikrodenetleyici yapısına eklenmesidir. Zamanlayıcı ve sayıcı donanım birimlerinin yaptığı işlemler yazılımsal olarak da yapılabilir. Fakat donanımsal olarak yapılan bu işler yazılımsal olarak yapıldığında, kesmeler bölümünde belirtildiği gibi yazılımın büyük çoğunluğu meşgul edilmiş olur. Zamanlayıcı/sayıcı birimleri ayrı ayrı ifade edilse dahi aynı birimlerdir. Aynı birim, kullanım şekline göre zamanlayıcı veya sayıcı olarak kullanılabilir.

Bir birimin **zamanlayıcı** olarak kullanılması için değer artışının düzenli (periyodik) olması gerekmektedir. **Sayıcı** olarak kullanım da ise değer artışının düzenli olması gerekliliği yoktur. Örneğin bir fabrikada banttan geçen şişelerin sayımının sensör vasıtası ile yapıldığını farz edelim. Bu işlemde şişelerin banttan geçiş hızı önemli

değildir. Bazen dakikada 100 şişe gelirken, bazen dakikada 150 şişe gelebilir. Önemli olan kaç adet şişe geçtiğinin tespitidir. Bu işlem için sayıcı birimi kullanılır. Aynı fabrikada şişelerin içine belli bir miktar sıvı konulduğunu farz edelim. Sıvının pompalandığı pompa borusunun çapı ve hızı belli olsun. Doldurulacak sıvı miktarının tespiti için sistemde pompa birimini belli bir süre aktif daha sonra pasif yapmak gereklidir. Bu işlem içinde mikrodenetleyicinin zamanlayıcı birimi kullanılabilir.

Zamanlayıcı ve sayıcı birimlerinin ikisinin kendi içeriklerinin artışı için bir sinyal kaynağına (clock) ihtiyacı vardır. Zamanlayıcı ile sayıcı arasındaki fark olarak şunu ifade edebiliriz. Zamanlayıcı, artışını sinyal kaynağı olarak mikrodenetleyicinin kristalinden sağlar. Yani denetleyicinin her komut çevriminde artacak şekilde çalışır. Sayıcı ise, sinyal kaynağı olarak harici girişlerden (dışarıdan) gelen sinyalleri kullanılır. Dolayısıyla zamanlayıcının artışı düzenli ve belli bir frekans değerinde olurken, sayıcının artışı düzenli olmak zorunda olmayan düzensiz frekans değerinde olmaktadır.

10.2. TIMER0 BİRİMİ

Timer0 birimi zamanlayıcı ve sayıcı olarak kullanılabilen bir donanım birimidir. Bu birimin özellikleri aşağıda verilmiştir.

8 bit Zamanlayıcı/Sayıcı kullanım,
Okuma ve Yazma yapılabilir,
8 bit'lik yazılım ile programlanabilen bölme oranı (prescaler)
Dâhili ve harici saat (clock) sinyali kaynağı seçme,
Timer0 değeri FFh'tan 00h'a (0-256) geçişinde taşma kesmesi,
Harici saat kaynağı sinyal tetikleme kenarı seçimi (edge select)

Timer0'ın kontrolü OPTION kaydedicisi ile sağlanır. Aşağıda OPTION kaydedicisi bit'lerinin açıklamaları verilmiştir.

OPTION KAYDEDİCİSİ

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Şekil-10.1. *OPTION kaydedicisi.*

Bit 7 **RBPU** = PORTB Pull-up yapma bit'i
0: Pull-up açık
1: Pull-up kapalı

Bit 6 **INTEDG** = RB0/INT pin'indeki kesme sinyalinin aktif olduğu kenarı seçme bit'i
0: Kesme düşen kenarda aktif
1: Kesme yükselen kenarda aktif

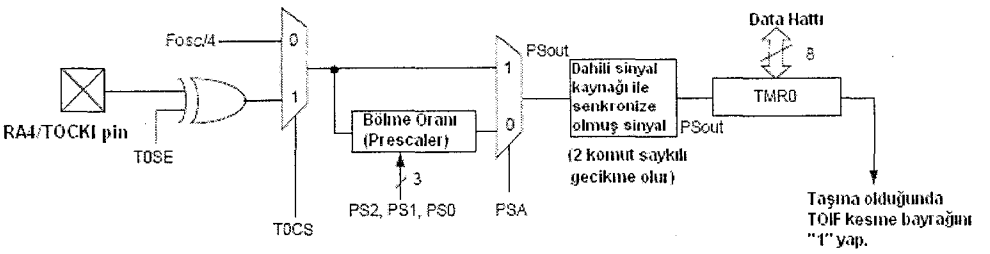
Bit 5 **TOCS** = TMR0 sinyal kaynağı seçme bit'i
0: Dâhili kaynak
1: Harici sinyal kaynağı

Bit 4 **TOSE** = TMR0 sinyal kaynağı kenar seçme bit'i
0: Düşen kenar tetiklemesi
1: Yükselen kenar tetiklemesi

Bit 3 **PSA** = Frekans bölücü seçme bit'i
0: Frekans bölme sayısı TMR0 için geçerli
1: Frekans bölme sayısı WDT için geçerli

Bit 2,1,0 **PS2, PS1, PS0** = Frekans bölme sayısı (Prescaler value)

Bit Değeri (PS2,PS1,PS0)	TMR0 ORANI	WDT ORANI
000	1:2	1.1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128



Şekil-10.2. *Timer0 blok diyagramı.*

Timer0 biriminin çalışmasını anlamak için OPTION kaydedicisi ve Timer0 blok diyagramına birlikte bakmak anlamayı kolaylaştıracaktır. Timer0 biriminin zamanlayıcı olarak çalışması için OPTION kaydedicisinin TOCS bit'i lojik-0 olmalıdır. Eğer Timer0 biriminin sayıcı olarak çalıştırılması isteniyorsa TOCS bit'i lojik-1 olmalıdır. Blok şemadan görüldüğü gibi Timer0 birimi ister dâhili kaynakla ister RA4/TOCKI pin'inden uygulanan harici saat sinyali kaynağından beslenir. Kaynak seçimine göre çalışma tipi (zamanlayıcı veya sayıcı) seçilmiş olur. Seçilen kaynaktan gelen saat sinyali direkt olarak kullanılmaz. Bu sinyal bölme oranı (prescaler) değeri ile çarpılarak elde edilen değer sinyal kaynağı olarak kullanılır. Bölme oranı değeri hem Timer0 için hem de Watchdog Timer için kullanılır. Bölme oranının hangi birim için kullanılacağı (Timer0 birimi için mi? yoksa WDT birimi için mi?) OPTION kaydedicisinin PSA bit'i ile belirlenir. Blok şemada da PSA bit'i görülmektedir. Normalde TMR0 kaydedicisi değeri başlangıçta sıfırdır. Programla data hattından TMR0 kaydedicisine istenen değer yüklenebilir. Böylece sayma işlemi istenen değerden başlatılabilir.

Timer0 birimi eğer sayıcı modunda çalıştırılırsa harici sinyal kaynağından gelen sinyalin hangi kenarında (yükselen veya düşen) artış olacağı OPTION kaydedicisinin TOSE bit'i ile belirlenir. Harici sinyal kaynağı kullanıldığında harici sinyal ile denetleyicinin dâhili saat sinyalinin faz senkronizasyonu gereklidir. Bu faz senkronizasyonu belli bir gecikmeye neden olur. Bu işlem için RA4/TOCKI girişinden uygulanan sinyalin $2T_{OSC}$ süresi kadar lojik-1 ve $2T_{OSC}$ süresi kadar lojik-0 olması gerekmektedir. Aynı zamanda her iki durumda da 20 ns'lik bir RC gecikmesi de $2T_{OSC}$ süresine eklenmelidir. Bu süreler sonunda timer0 birimi artmaya başlayabilir.

Timer0 biriminin kullanımı için anlatılan kaydedici bit'leri gerekli konuma (1 veya 0) alınmalıdır. CCS C derleyicisinde bu işlemler için anlatılan kaydedici bit'leri ile işlem yapmaya gerek yoktur. Hazır zamanlayıcı/sayıcı fonksiyonları ile kurulum işlemleri yapılır. Anlatılan bilgiler fonksiyonun kullanımına daha fazla hâkimiyet ve anlaşılırlık katmak içindir.

10.2.1. SETUP_TIMER_0 () Fonksiyonu

Timer0 zamanlayıcı ayarlarını yapmaya yarayan bir fonksiyondur.

setup_timer_0(*mod*);

Fonksiyondaki "*mod*" kısmına aşağıda belirtilen sabit tanımlamalardan biri veya birden fazlası yazılabilir. İki sabit arasında "|" operatörü konulur. Bu fonksiyonda kullanılacak sabitler denetleyicinin başlık dosyasında (Örneğin, 16F877.h gibi) yazılıdır.

Kesme Aktif Kenar Ayarı Sabitleri;

"RTCC_INTERNAL" = Timer0 clock kaynağının denetleyici çalışma frekansı olacağını bildirir.

"RTCC_EXT_L_TO_H" =Timer0 clock kaynağının dış çevreden alınacağını ve sinyalin her yükselen kenarında tetikleme olacağını bildirir.

"RTCC_EXT_H_TO_L" = Timer0 clock kaynağının dış çevreden alınacağını ve sinyalin her düşen kenarında tetikleme olacağını bildirir.

Bölme Oranı (Prescaler) Sabitleri;

"RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256"

Sabitin sonundaki rakamlar bölme oranını belirtir.

Sadece PIC18XXX ürünleri için RTCC_OFF , RTCC_8_BIT mod komutları da mevcuttur.

Örnek;

```
setup_timer_0 ( RTCC_DIV_4 | RTCC_EXT_L_TO_H );
```

komutu ile Timer0'ın harici kaynaktan besleneceği (sayıcı olarak kullanılacağı), harici kaynak sinyalinin her yükselen kenarında artış olacağı ve bölme oranının 4 olduğu belirtilmiş olur.

10.2.2. SET_TIMER0 ve SET_RTCC() Fonksiyonu

SET_TIMER0 ve SET_RTCC () fonksiyonları aynı anlamdadır. İki fonksiyonda Timer0 biriminin saymaya başlama değerini belirlemek için kullanılırlar. Tüm sayıcılar yukarı doğru sayar ve maksimum değere ulaşınca sıfıra döner ve tekrar yukarı saymaya başlar.

set_timer0(*değer*); veya **set_rtcc(*değer*);**

Fonksiyonda "*değer*" kısmına Timer0 biriminin saymaya başlama değeri yazılır. Timer0 birimi 8 bit'lik olduğu için buraya yazılacak değer 0 ile 255 arasında olmalıdır.
Örnek;

```
set_timer0(45);
```

komutu ile Timer0 birimi saymaya 45'den itibaren başlar.

10.2.3. Timer0 Kesmesi (#INT_TIMER0)

Timer0 biriminin gerekli kurulumu yapıldıktan sonra istenirse taşma kesmesi de oluşturulabilir. 9. bölümde anlatılan INTCON kaydedicisinin TOIE bit'i ile timer0 taşma kesmesi oluşturulup oluşturulmayacağı belirlenir. Bu iş için CCS C'de kesmeler bölümünde anlatılan aynı mantıkla #INT_TIMER0 komutu ile kesme fonksiyonu oluşturulur. Burada dikkat edilmesi gereken nokta kesme fonksiyonu içinde mutlaka set_timer0() komutu ile programda set edilen değerin tekrar kesme fonksiyonu içinde de belirtilmesi gerekliliğidir. Çünkü program kesme fonksiyonuna gittiğinde, TMR0 kaydedicisi içeriği sıfırlanmış demektir. Timer0 biriminin tekrar başlangıçta istenen değer ile saymaya başlanması isteniyorsa kesme fonksiyonunda mutlaka tekrardan set_timer0() komutu ile set değeri belirtilmelidir.

Timer0 kaydedici için oluşturulacak kesmenin oluşma süresi aşağıdaki formül ile hesaplanabilir.

Kesme Süresi T_{osc}x(Bölme Oranı)x(256 TMR0'a yazılan değer)

T_{osc} periyodu ; $T_{osc} \quad \frac{1}{f_{osc}}$ formülünden bulunur.

f_{osc} değeri bilindiği gibi PIC denetleyiciye bağlanan osilatör frekansının 4'e bölümü ile elde edilir.

$$f_{osc} \quad \frac{\text{Denetleyici Osilatör Frekansı}}{4}$$

10.2.4. GET_TIMERx () Fonksiyonu

Bu fonksiyon komutu istenen zamanlayıcı ve sayıcının kaydedici (TMR0, TMR1 vb. kaydedicilerin) değerini bize verir. GET_TIMER0 komutu ile GET_RTCC() aynı anlamdadır. get_timerx () komutunda x yerine istenen zamanlayıcı/sayıcı biriminin numarası yazılır. Bu komutta dikkat edilmesi gereken nokta Timer0 ve Timer2 birimleri 8 bit'lik olduğundan bu komut kullanıldığında geri dönüş değerinin kaydedileceği değişken 8 bit'lik olmalıdır. Örneğin Timer1 birimi 16 bit olduğundan bu birimin kaydedicisi içeriği bu komutla okunmak istendiğinde geri dönüş değerinin kaydedileceği değişken 16 bit'lik olmalıdır.

değer = get_timer0( ), *değer* = get_rtcc( ), *değer* = get_timer1( ),
değer = get_timer2( ), *değer* = get_timer3( )

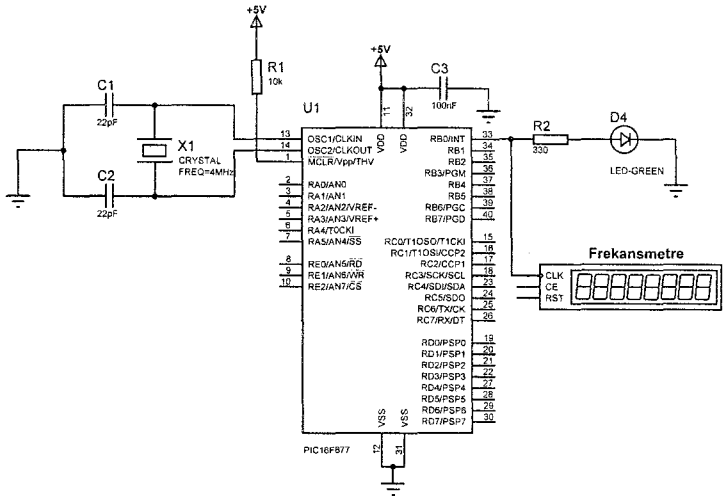
Örnek;

```
x = get_timer2( );   // Timer2 biriminin TMR2 kaydedici değeri x
                     // değişkenine atanır.
```

10.2.5. Timer0 Uygulaması-1

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada Timer0 birimi zamanlayıcı olarak kullanılmıştır. Timer0 kesmesi her 0,050176 saniyede bir kesme oluşturacak şekilde ayarlanmıştır. Kesme fonksiyonunda ise her 10 kesme oluştuğunda yani yaklaşık her 0.5 saniyede bir RB0 çıkışına bağlı LED'in yanması, diğer 10 kesme oluştuğunda ise (20. kesmede) yani 0.5 saniyede sonra RB0 çıkışına bağlı LED'in sönmesi sağlanmıştır. Böylece yaklaşık 2 Hz civarında bir çıkış sinyali elde edilmiştir.



Şekil-10.3. *Timer0 uygulaması-1 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Timer0 Uygulaması-1
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>   // Kullanılacak denetleyicinin başlık
                      //dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

int i=0;  // Tamsayı tipinde değişken tanımlanıyor

//****************** Timer0 Kesmesi ****************************
#int_timer0  // Timer0 kesmesi
void  timer0_kesme ()   // Kesme fonksiyonu ismi
{
    set_timer0(60);   // TMR0 değeri belirleniyor
    i++;   // i değeri 1 arttırılıyor

    if (i==10)  // i değeri 10'a eşitse RB0 çıkışı lojik-1 olsun
       output_high(pin_b0);

    if (i==20)  // i değeri 20'ye eşitse RB0 çıkışı lojik-1 olsun ve
                // i değeri sıfırlansın
       {
          output_low(pin_b0);
          i=0;
       }
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı


   set_tris_b(0x00);   // B port'u komple çıkış
   output_b(0x00);     // B port'u çıkışı ilk anda sıfırlanıyor

   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256); // Timer0 ayarları
                                                 //yapılıyor
```

```
   set_timer0(60);    // TMR0 değeri belirleniyor

   enable_interrupts(INT_timer0);// int_timer0 kesmesini aktif yapar
   enable_interrupts(GLOBAL);     // Aktif edilen kesmelere izin ver

   while(1);    // Sonsuz döngü
 }
//*****************************************************************
```

Ana fonksiyonda `setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256);` komutu ile timer0 sinyal kaynağının dâhili olduğu (RTCC_INTERNAL kısmı ile) ve bölme oranının 256 olduğu (RTCC_DIV_256 kısmı ile) belirtilmiştir. Ardından `set_timer0(60);` komutu ile de Timer0 biriminin saymaya başlama değeri 60 olarak belirlenmiştir (TMR0 kaydedicisinin içine 60 değeri yüklenmiştir). `enable_interrupts(INT_timer0);` komutu ile timer0 kesmesi aktif yapılmış ve `enable_interrupts(GLOBAL);` komutu ile de tüm aktif edilen kesmelere izin verilmiştir. Bu şekli ile kesme fonksiyonu süresini hesaplarsak şu değerler bulunur.

$$f_{osc} \quad \frac{\text{Denetleyici Osilatör Frekansı}}{4} \quad \frac{4\text{Mhz}}{4} \quad 1\text{Mhz}$$

Kesme Süresi T_{osc}x(Bölme Oranı)x(256 TMR0'a yazılan değer)

$= (1/f_{osc})$x256x(256-60)=(1/1Mhz)x256x196=0,1 snx256x196

=50176 sn=50,176msn=0,050176sn

Kesme fonksiyonu `#int_timer0` ve `void timer0_kesme ()` komutları ile tanıtılmıştır. Kesme fonksiyonunda her kesme oluştuğunda i değeri `i++;` komutu ile 1 arttırılmıştır. `if (i==10)` komutu ile i değerinin 10'a eşit olup olmadığı kontrol edilmiştir. i değeri 10 olduğunda yani 10 kez kesme meydana geldiğinde `output_high(pin_b0);` komutu ile RB0 çıkışı lojik-1 yapılarak LED yakılmıştır. `if(i==20)` komutu ile i değerinin 20'ye eşit olup olmadığı kontrol edilmiş ve i değeri 20 olduğunda yani 20 kez kesme meydana geldiğinde `output_low(pin_b0);` komutu ile RB0 çıkışı lojik-0 yapılarak LED söndürülmüş ve aynı kontrol döngüsünde `i=0;` komutu ile de i değeri sıfırlanmıştır. Burada dikkat edilmesi gereken nokta, programda belirtilen timer0 başlangıç değeri kesme fonksiyonu başında yine `set_timer0(60);` komutu ile belirtilmiştir. Çünkü her kesme meydana geldiğinde TMR0 kaydedicisi sıfırlanır. Bu kaydedicinin yine istenen değerden başlayarak sayma yapması için kesme fonksiyonunda bu işlem belirtilmelidir. Kesme fonksiyonunda her 10 kesmeden sonra LED yakıldığına göre aşağıdaki hesaplamalar yapılabilir. Hesaplamalardan görüleceği üzere LED 0.5 sn yanmakta ve 0,5 sn sönmektedir. Çıkış ucu bir frekans metreye bağlanırsa çıkış sinyali frekansı ölçülebilir.

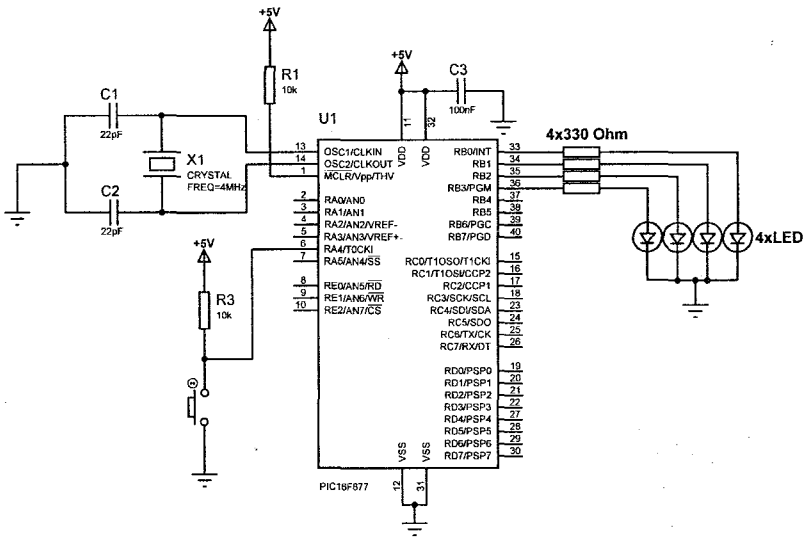
LED Yanma Süresi=10xKesme süresi=10x0,050176sn=0,50176sn

$$\text{Çıkış Frekansı}=\frac{1}{\text{LED Yanma Süresi}}=\frac{1}{0{,}50176} \quad 1{,}993\text{Hz} \quad 2\text{Hz}$$

10.2.6. Timer0 Uygulaması-2

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada Timer0 birimi sayıcı olarak kullanılmıştır. RA4/TOCKI girişine bir buton bağlanmış ve Timer0 harici sinyal girişi buton vasıtası ile sağlanmıştır. RA4/TOCKI girişi normalde yüksek seviyede yani lojik-1'dir. Program vasıtası ile harici sinyalin her düşen kenarında tetikleme alınması sağlanmıştır. Bölme oranı 2, saymaya başlama değeri de 254 olarak ayarlanmıştır. Timer0 birimi 256-254=2 sayma yapacaktır. Bu bu sayma aynı zamanda bölme oranı ile çarpıldıktan sonra elde edilen değer sonucunda kesme meydana gelecektir. Bölme oranı 2 olarak programlandığından 2x2=4 kez butona basılınca bir kesme meydana gelecektir. RB0, RB1, RB2 ve RB3 çıkışlarına LED bağlanmıştır. Her kesme fonksiyonunda çıkış değeri 1 arttırılmaktadır. LED'lerde görünen çıkış değeri 15 olunca çıkış değeri sıfırlanmaktadır. Devrenizi denerken buton arklarından dolayı butona 4 kez değil de örneğin 3 veya 2 kez bastığınız zamanlarda da artış olması buton arklarından dolayıdır.



Şekil-10.4. *Timer0 uygulaması-2 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Timer0 Uygulaması-2
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>  // Kullanılacak denetleyiçinin başlık
                     //dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
```

```
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           //osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

int sayi=0; // Tamsayı tipinde değişken tanımlanıyor

//****************** Timer0 Kesmesi ****************************
#int_timer0  // Timer0 kesmesi
void  timer0_kesme ()    // Kesme fonksiyonu ismi
{
   set_timer0(254);   // TMR0 değeri belirleniyor

   sayi++; // sayi değeri 1 arttrılıyor
   output_b(sayi);   // sayi değeri B port'una gönderiliyor

   if (sayi==15)   // sayi değeri 15 olunca sıfırla
      sayi=0;
}
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x10); // RA4 pin'i giriş
   set_tris_b(0x00); // B port'u k mple çıkış

   output_b(0x00);     // B port'u çıkışı ilk anda sıfırlanıyor

   setup_timer_0(RTCC_EXT_H_TO_L | RTCC_DIV_2); // Timer0 ayarları
                                                // yapılıyor
   set_timer0(254);   // TMR0 değeri belirleniyor

   enable_interrupts(INT_timer0);// int_timer0 kesmesini aktif yapar
   enable_interrupts(GLOBAL);    // Aktif edilen kesmelere izin ver

   while(1);   // Sonsuz döngü
 }
//*****************************************************************
```

Programda `setup_timer_0(RTCC_EXT_L_TO_H | RTCC_DIV_2);` komutu ile Timer0 biriminin harici kaynaktan yani RA4/TOCKI girişine uygulanan sinyal ile besleneceği belirtilmiştir. Komutta `RTCC_EXT_H_TO_L` kısmı ile harici kaynak sinyalinin düşen kenarında tetikleme alınacağı belirtilmiştir. Komuttaki `RTCC_DIV_2` kısmı ile de bölme değerinin 2 olacağı belirtilmiştir. Saymaya başlama değeri ise

`set_timer0(254);` komutu ile 254 olarak belirlenmiştir. Timer0 kesmesinin oluşması kaç kere butona basmak gerekli olduğunu hesaplarsak;

Kesme Oluşma Sayısı (256 254)x(Bölme değeri) 2x2 4 olarak bulunur.

Her kesme oluştuğunda kesme fonksiyonunda `sayi++;` komutu ile "sayi" değişkeni 1 arttırılmış ve `output_b(sayi);` komutu ile de B port'una gönderilmiştir. `if (sayi==15)` komutu ile sayi değişkeni değerinin 15'e eşit olup olmadığı kontrol edilmiş ve eşit olduğunda `sayi=0;` komutu ile sıfırlanması sağlanmıştır.

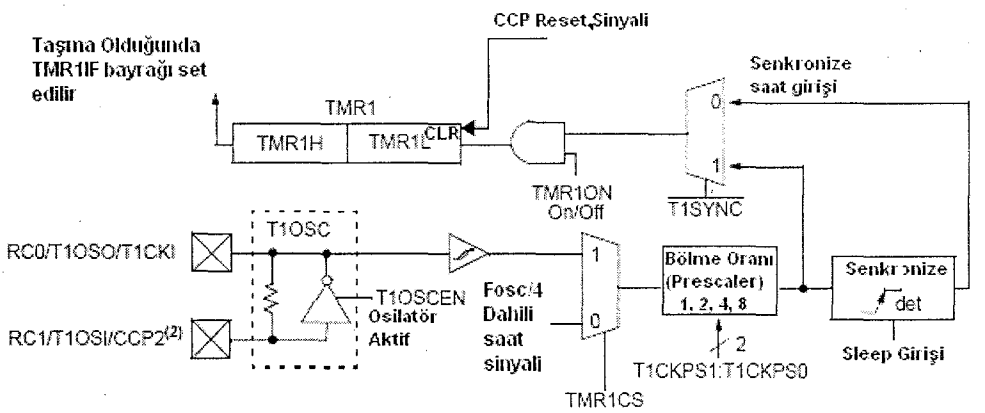
10.3. TIMER1 BİRİMİ

Timer1 birimi zamanlayıcı ve sayıcı olarak kullanılabilen bir donanım birimidir. Bu birimin özellikleri aşağıda verilmiştir.

16 bit Zamanlayıcı/Sayıcı kullanım,
Okuma ve Yazma yapılabilir,
Dahili ve harici saat (clock) sinyali kaynağı seçme,
Timer1 değeri FFFFh'tan 0000h'a (0-256) geçişinde taşma kesmesi.

Timer1 birimi içinde iki adet 8 bit'lik kaydedici bulundurur. Bunlar TMR1H ve TMR2H kaydedicilerdir. Timer1 ile 0 - 65536 (0000h - FFFFh) arasında istenen sayıdan başlanarak sayma işlemi yapılabilir. Timer1 birimi 3 değişik modda kullanılabilir.

Senkron zamanlayıcı
Senkron Sayıcı
Asenkron Sayıcı



Şekil-10.5. *Timer1 birimi blok diyagramı.*

Zamanlayıcı modunda Timer1 her komut çevriminde bir artar. Sayıcı modunda ise Timer1, RC0/T1OSO/**T1CKI** pin'inden gelen saat sinyalinin **her yükselen kenarında bir artar.** Timer1 aynı zamanda dâhili bir sıfırlama (reset) kaynağına sahiptir. Bu sıfırlama sinyali CCP modülünden gelen sıfırlama sinyalidir. CCP modülünden gelen sinyal ile Timer1 içeriği sıfırlanabilir. Diğer sıfırlama metotları TMR1H ve TMR2H kaydedici içeriklerini sıfırlamaz.

Anlatılan tüm Timer1 sayıcı/zamanlayıcısı özelliklerini kontrol eden kaydedici T1CON kaydedicisidir.

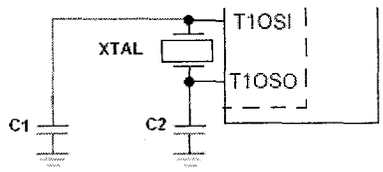
T1CON KAYDEDİCİSİ

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

Şekil-10.6. *T1CON kaydedicisi.*

Bit 7-6 = Kullanılmayan bit'ler. Okunurken bu bit'ler "0" bilgisi olarak okunur.
Bit 5-4 **T1CKPS1-T1CKPS0** = Timer1 bölme (prescaler) oranı
11 = 1:8
10 = 1:4
01 = 1:2
00 = 1:1
Bit 3 **T1OSCEN** = Timer1 osilatör kontrol bit'i
1:Osilatör açık
0:Osilatör kapalı
Bit 2 **$\overline{\text{T1SYNC}}$** = Timer1 dış clock sinyali senkronizasyon kontrol bit'i
1:Harici clock sinyali girişi ile senkronize yok
0:Harici clock sinyali girişi ile senkronize
Bit 1 **TMR1CS** = Timer1 clock kaynağı seçme bit'i
1:RC0/T1OSO/T1CKI pin'inden dış clock sinyali
0:Dâhili clock sinyali (F_{osc} /4)
Bit 0 **TMR1ON** = Timer1 açma-kapama bit'i
1:Timer1 açık
0:Timer1 kapalı

Timer1 biriminin ayrıca T1OSI (giriş-input) ve T1OSO (çıkış-output) pin'leri arasına harici osilatör bağlanabilir. Bu harici osilatör 32 Khz ile 200 Khz arasında olabilen düşük güçlü kristal ve kondansatörlü osilatör olabilir. Bu sayede uyku modunda bile Timer1 birimi çalışmaya devam eder. Bu girişler aslında çok daha ideal bir çalışma frekansı olan 32 Khz için tasarlanmıştır. Osilatör kristaline göre kullanılması gereken kondansatörler Tablo-10.1'de verilmiştir. Osilatörün ilk çalışmada kararlı bir sinyal üretmeye başlaması için gerekli süre program komutları ile verilmelidir.



Kristal Frekansı	C1	C2
32 Khz	33 pF	33 pF
100 Khz	33 pF	33 pF
200 Khz	15 pF	15 pF

Şekil-10.7. *Timer1 harici osilatör bağlantısı*

Tablo-10.1. *Timer1 osilatörü kondansatör değerleri*

10.3.1. Timer1 Kesmesi (#INT_TIMER1)

Timer1 istenen değerden saymaya başlar ve 65536 sayısına ulaşınca Timer1 kesme taşması meydana gelir. Timer1 kesmesine izin verme bit'i PIE1 kaydedicisinin 0.bit'i olan TMR1IE bit'idir. **Timer1 kesmesi #int_timer1 komutu ile ifade edilir. Dikkat edilmesi gereken bir nokta vardır. Kesme fonksiyonu içinde mutlaka set_timer1() komutu ile programda set edilen değerin tekrar kesme fonksiyonu içinde de belirtilmesi gerekliliğidir.** Çünkü program kesme fonksiyonuna gittiğinde, Timer1 taşmış yani 0000h değerine gelmiş demektir. Timer1 biriminin tekrar başlangıçta istenen değer ile saymaya başlanması isteniyorsa kesme fonksiyonunda mutlaka tekrardan set_timer1() komutu ile set değeri belirtilmelidir.

Timer1 kesme süresi aşağıdaki formül ile hesaplanabilir.

Kesme Süresi T_{osc}x(Bölme Oranı)x(65536 TMR1'e yazılan değer)

T_{osc} periyodu ise hatırlanacağı gibi; $T_{osc} \quad \frac{1}{f_{osc}}$ formülünden hesaplandığını unutmayalım.

10.3.2. SETUP_TIMER1() Fonksiyonu

Bu komut Timer1 zamanlayıcı ayarlarını yapmaya yarayan bir fonksiyondur.

setup_timer_1(***mod*** **);**

Fonksiyondaki "*mod*" kısmına aşağıda belirtilen sabit tanımlamalardan biri veya birden fazlası yazılabilir. İki sabit arasında "|" operatörü konulur. Bu fonksiyonda kullanılacak sabitler denetleyicinin başlık dosyasında (Örneğin, 16F876.h) yazılıdır.

Timer1 Kontrol Sabitleri;
"T1_DISABLED" =Timer1'i kapat demek),
"T1_INTERNAL" = Timer1'ın clock sinyalinin kaynağının denetleyici frekansı olacağını belirtir.

"T1_EXTERNAL"=Timer1'ın clock sinyalinin kaynağının harici olacağını belirtir.
"T1_EXTERNAL_SYNC" =Timer1 harici sinyal kaynağıyla senkronize olacağını bildirir.
"T1_CLK_OUT" =Timer1 osilatörünün kullanılacağını bildirir.

Timer1 Zamanlayıcı Bölme Oranları (Prescaler) Sabitleri:
"T1_DIV_BY_1, T1_DIV_BY_2, T1_DIV_BY_4, T1_DIV_BY_8"
Sabitin sonundaki rakamlar bölme oranını belirtir.

Örnek;

```
setup_timer_1( T1_EXTERNAL | T1_DIV_BY_4 );
```

Örnekte, Timer1 birimi, harici kaynaktan saat sinyali alacak ve bölme oranı 4 olacak şekilde kurulmuştur.

10.3.3. SET_TIMER1() Fonksiyonu

Bu komut Timer1 biriminin saymaya başlama değerini belirlemek için kullanılırlar. Tüm sayıcılar yukarı doğru sayar ve maksimum değere ulaşınca sıfıra döner ve tekrar yukarı saymaya başlar.

set_timer1(*değer*);

Fonksiyonda "*değer*" kısmına Timer1 biriminin saymaya başlama değeri yazılır. Timer1 birimi 16 bit'lik olduğu için buraya yazılacak değer 0 ile 65536 arasında olmalıdır.

Örnek;

```
set_timer1(764);
```

komutu ile Timer1 birimi saymaya 764'den itibaren başlar.

10.3.4. Timer1 Uygulaması-1

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada Timer1 birimi zamanlayıcı olarak kullanılmıştır. Timer1 kesmesi her 20 mili saniyede bir kesme oluşturacak şekilde ayarlanmıştır. Kesme fonksiyonunda ise her 50 kesme oluştuğunda yani yaklaşık her 1 saniyede bir RB0 çıkışına bağlı LED'in yanması, diğer 50 kesmede de yani 1 saniyede de RB0 çıkışına bağlı LED'in sönmesi sağlanmıştır. Kullanılan devre Şekil-10.3'te verilen devre ile aynıdır.

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Timer1 Uygulaması-1
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>   // Kullanılacak denetleyicinin başlık
                      //dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

int i=0;  // Tamsayı tipinde değişken tanımlanıyor

//****************** Timer1 Kesmesi ****************************
#int_timer1  // Timer1 kesmesi
void  timer1_kesme ()   // Kesme fonksiyonu ismi
{
    set_timer1(63016);   // TMR1 değeri belirleniyor
    i++;   // i değeri 1 arttırılıyor

    if (i==50)  // i değeri 10'a eşitse RB0 çıkışı lojik-1 olsun
       output_high(pin_b0);
    if (i==100)   // i değeri 20'ye eşitse RB0 çıkışı lojik-1 olsun
                  // ve i değeri sıfırlansın
    {
          output_low(pin_b0);
          i=0;
    }
}
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00);   // B port'u komple çıkış
   output_b(0x00);     // B port'u çıkışı ilk anda sıfırlanıyor

   setup_timer_1(T1_INTERNAL | T1_DIV_BY_8); // Timer0 ayarları
                                             //yapılıyor
   set_timer1(63036);   // TMR1 değeri belirleniyor

   enable_interrupts(int_timer1); //Timer1 kesmesini aktif yapar
   enable_interrupts(GLOBAL);     // Aktif edilen kesmelere izin ver
```

```
  while(1);    // Sonsuz döngü
 }
//*****************************************************************
```

Programda ilk dikkat edeceğiniz nokta öneceki her programımızda, ana fonksiyon başında denetleyicide kullanmadığımız fonksiyonları, birimleri devre dışı bırakıyorduk. Bu programda Timer1 birimini kullandığımız için öceki tüm programlarda kullandığımız `setup_timer_1(T1_DISABLED);` komutunun bu programda yer almadığına dikkat edin. Bu komutu kullanmış olsanız dahi programda yine `setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);` komutu kullanıldığından en son Timer1 komutu geçerli olacağından programınız yine sorunsuz çalışacaktı.

Ana fonksiyonda `setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);`komutu ile timer1 sinyal kaynağının dâhili olduğu (T1_INTERNAL kısmı ile) ve bölme oranının 8 olduğu (T1_DIV_BY_8 kısmı ile) belirtilmiştir. Ardından `set_timer1(63036);` komutu ile de Timer1 biriminin saymaya başlama değeri 63036 olarak belirlenmiştir `enable_interrupts(INT_timer1);` komutu ile timer1 kesmesi aktif yapılmış ve `enable_interrupts(GLOBAL);` komutu ile de tüm aktif edilen kesmelere izin verilmiştir. Bu şekli ile kesme fonksiyonu süresini hesaplarsak şu değerler bulunur.

$$f_{osc} \quad \frac{\text{Denetleyici Osilatör Frekansı}}{4} \quad \frac{4\text{Mhz}}{4} \quad 1\text{Mhz}$$

Kesme Süresi T_{osc}x(Bölme Oranı)x(65536 TMR1'e yazılan değer)

= $(1/f_{osc})$x8x(65536-63036)=(1/1Mhz)x8x2500=1 snx8x2500

=20000 sn=20msn=0,02sn

Kesme fonksiyonu `#int_timer1` ve `void timer1_kesme ()` komutları ile tanıtılmıştır. Kesme fonksiyonunda her kesme oluştuğunda i değeri `i++;` komutu ile 1 arttırılmıştır. `if (i==50)` komutu ile i değerinin 100'e eşit olup olmadığı kontrol edilmiştir. i değeri 50 olduğunda yani 50 kez kesme meydana geldiğinde `output_high(pin_b0);` komutu ile RB0 çıkışı lojik-1 yapılarak LED yakılmıştır. `if(i==100)` komutu ile i değerinin 100'e eşit olup olmadığı kontrol edilmiş ve i değeri 100 olduğunda, yani 100 kez kesme meydana geldiğinde `output_low(pin_b0);` komutu ile RB0 çıkışı lojik-0 yapılarak LED söndürülmüş ve aynı kontrol döngüsünde `i=0;` komutu ile de i değeri sıfırlanmıştır. Burada dikkat edilmesi gereken nokta, programda belirtilen timer1 başlangıç değeri (set_timer1() ile verilen) kesme fonksiyonu başında yine belirtilmiştir. Çünkü her kesme meydana geldiğinde TMR1 kaydedicisi sıfırlanır. Bu kaydedicinin yine istenen değerden başlayarak sayma yapması için kesme fonksiyonunda bu işlem belirtilmelidir. Kesme fonksiyonunda her 50 kesmeden sonra LED yakıldığına göre aşağıdaki hesaplamalar yapılabilir. Hesaplamalardan görüleceği üzere LED 1 sn yanmakta ve 1 sn sönmektedir. Çıkış ucu bir frekans metreye bağlanırsa çıkış sinyali frekansı ölçülebilir.

LED Yanma Süresi=50xKesme süresi=50x0,02sn=1sn

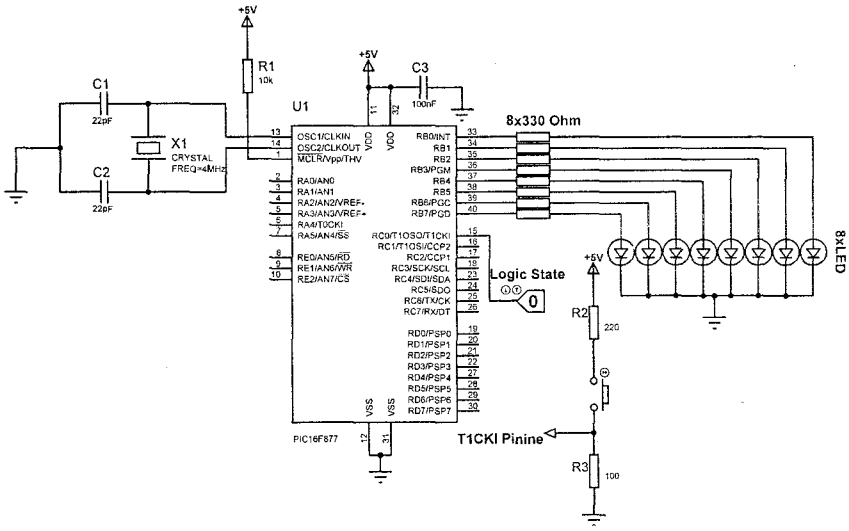
$$\text{Çıkış Frekansı}=\frac{1}{\text{LED Yanma Süresi}}=\frac{1}{1}\ \text{1Hz}$$

10.3.5. Timer1 Uygulaması-2

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada da Timer1 birimi sayıcı olarak (counter) kullanılmıştır. Bunun için RC0/T1OSO/T1CKI pin'ine bir buton bağlanmış ve butona basmak suretiyle elde edilen sinyal harici giriş sinyal kaynağı olarak kullanılmıştır. Programda bölme oranı 2, saymaya başlama değeri de 65530 olarak ayarlanmıştır. Timer1 birimi 65536-65530=6 sayma yapacaktır. Bu bu sayma aynı zamanda bölme oranı ile çarpıldıktan sonra elde edilen değer sonucunda kesme meydana gelecektir. Bölme oranı 2 olarak programlandığından 6x2=12 kez butona basılınca yani 12 kez yükselen kenar tetiklemesi geldikten sonra bir kesme meydana gelecektir. B port'u çıkışlarına LED bağlanmıştır. Her kesme fonksiyonunda çıkış değeri 1 arttırılmaktadır. LED'lerde görünen çıkış değeri FFFFh olunca çıkış değeri sıfırlanmaktadır.

Devrenizi denerken buton arklarından dolayı butona 12 kez değil de daha az sayıda bastığınız zamanlarda da LED'lerde artış olması buton arklarından dolayıdır. Devre şemasında buton bağlantısının yanında proteus programında devrenizi denerken daha net olarak sonucu görmeniz için RC0/T1OSO/T1CKI pin'ine Logic State (mantıksal durum) ucu bağlanmıştır. Bu sayede tam olarak 12 adet yükselen kenardan (lojik-0'dan lojik-1'e geçiş) sonra kesme oluştuğu görülebilir.



Şekil-10.8. *Timer1 uygulaması-2 devre şeması.*

Devrenizi PIC/PROG DEKA 16F87x deneme kartınızda denemek için deneme kartının sol kenarında yer alan çıkış uçlarından Vcc pin'ine zil telinin bir ucunu yerleştiriniz. Zil telinin diğer ucunu elinizle PIC16F877'nin RC0 ucuna değdirip çekin. Böylece el ile kendiniz RC0 ucuna yükselen kenar tetikleme sinyali vermiş olursunuz. Belli sayıda elinizi değdirip çektikçe LED'lerde artışı görebilirsiniz.

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Timer1 Uygulaması-2
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.

// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

unsigned int kesme=0; //Tamsayı tipinde işaretsiz değişken
                      //tanımlanıyor

//****************** Timer1 Kesmesi ****************************
#int_timer1  // Timer1 kesmesi
void  timer1_kesme ()    // Kesme fonksiyonu ismi
{
   set_timer1(65530); // TMR1 değeri belirleniyor

   kesme++; // kesme değeri 1 arttırılıyor
   output_b(kesme); // kesme değeri B port'una aktarılıyor

   if (kesme==255)   // Eğer kesme değeri 255 olursa kesme değişkeni
                     // içeriğini sıfırla
      kesme=0;
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00);    // B port'u komple çıkış
```

```
    set_tris_c(0x01);    // RC0 pin'i giriş

    output_b(0x00);      // B port'u çıkışı ilk anda sıfırlanıyor

    setup_timer_1(T1_EXTERNAL | T1_DIV_BY_2); // Timer1 ayarları
                                               // yapılıyor
    set_timer1(65530);   // TMR1 değeri belirleniyor
    enable_interrupts(INT_timer1);// int_timer1 kesmesini aktif yapar
    enable_interrupts(GLOBAL);    // Aktif edilen kesmelere izin ver

    while(1); // Sonsuz döngü
  }
//*****************************************************************
```

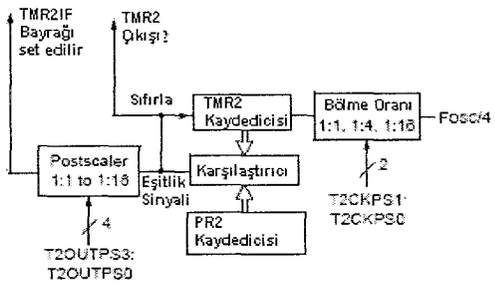
Programda `setup_timer_1(T1_EXTERNAL | T1_DIV_BY_2);` komutundaki `T1_EXTERNAL` kısmı ile Timer1 biriminin RC0/T1OSO/T1CKI girişine uygulanan sinyal ile besleneceği belirtilmiştir. Komuttaki `T1_DIV_BY_2` kısmı ile de bölme değerinin 2 olacağı belirtilmiştir. Saymaya başlama değeri ise `set_timer1(65530);` komutu ile 65530 olarak belirlenmiştir. Timer1 kesmesinin oluşması kaç kere butona basmak gerekli olduğunu hesaplarsak;

Kesme Oluşma Sayısı (65536 65530)x(Bölme değeri) 6x2 12 olarak bulunur.

Her kesme oluştuğunda kesme fonksiyonunda `kesme++;` komutu ile "kesme" değişkeni 1 arttırılmış ve `output_b(kesme);` komutu ile de B port'una gönderilmiştir. `if (kesme==255)` komutu ile "kesme" değişkeni değerinin 255'e eşit olup olmadığı kontrol edilmiş ve eşit olduğunda `kesme=0;` komutu ile sıfırlanması sağlanmıştır.

10.4. TIMER2 BİRİMİ

Timer2 birimi diğer Timer0 ve Timer1 birimlerinden farklı olarak sadece zamanlayıcı olarak kullanılan 8 bit'lik bir birimdir. Timer2 birimi 8 bit'lik bölme oranı (prescaler) ve diğer zamanlayıcı/sayıcı birimlerinde olmayan 8 bit'lik postscaler değerine sahiptir. Timer2 aynı zamanda denetleyicide bulunan CCP modülündeki PWM birimi için sinyal üretmede kullanılan zamanlayıcıdır.



Şekil-10.9. *Timer2 birimi blok diyagramı.*

Timer2 biriminin blok şemasında görüleceği gibi Timer2 zamanlayıcısı için denetleyicinin dâhili saat sinyali ($f_{osc}/4$) kullanılır. Bu sinyal bölme oranı (prescaler) ile çarpılarak kaynak saat sinyalini oluşturur. 8 bit'lik TMR2 zamanlayıcı kaydedicisi içindeki değer denetleyici frekansı ile bölme oranı ile hesaplanan frekansta artar. PR2 8 bit'lik yazılabilir ve okunabilir bir periyot kaydedicisidir. TMR2 kaydedicisi içindeki değer 00h'tan başlayarak sayar ve PR2 kaydedicisi içine yazılan değere eşit olduğunda (Eşitlik tespitini, blok şemada görülen karşılaştırıcı -comparator- birimi yapar) bir eşitlik sinyali üretilir ve TMR2 kaydedicisi sıfırlanır. Eşitlik sinyali sonucunda PR2 kaydedicisi de FFh değerini alır. **Postscaler** değeri meydana gelen bu eşitlik sinyallerinden kaç defa meydana geldikten sonra Timer2 kesmesi sinyali oluşacağını belirler. Örneğin postscaler değeri 2 ise, 2 defa TMR2 kaydedicisi değeri PR2 değerine eşit olunca Timer2 kesmesi oluşur. Tüm bu işlemler **T2CON** kaydedicisi ile belirlenir. **Bölme oranı (prescaler) ve Postscale değerleri POR, MCLR, WDT vb. sıfırlama (reset) sinyallerinden biri oluştuğunda veya TMR2 kaydedicisine veri yazma işlemi ile T2CON kaydedicisine veri yazma işlemlerinden biri gerçekleştiğinde sıfırlanır.**

T2CON KAYDEDİCİSİ

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Şekil-10.10. *T2CON kaydedicisi.*

Bit 7 = Kullanılmayan bit. Okunurken bu bit "0" bilgisi olarak okunur.

Bit 6-3 **TOUTPS3:TOUTPS0** = Timer2 Postscale değeri seçme bit'leri
0000: 1:1
0001: 1:2
0002: 1:3
0003: 1:4
...
...
1111 = 1:16

Bit 2 **TMR2ON** = Timer2 açma-kapama bit'i
1: Timer2 açık
0: Timer2 kapalı

Bit 1-0 **T2CKPS1-T2CKPS2** = Timer2 bölme (prescaler) değeri seçme bit'leri
00 = Bölme değeri 1
01 = Bölme değeri 4
1x = Bölme değeri 16

R= Okunabilir bit, **W=** İlgili bit'e bilgi yazılabilir, **U=** Kullanılmayan bit, **-0=** İlk enerji geldiğinde "0" olarak okunur, **x=** Bit değerinin 1 veya 0 olması önemli değil, anlamındadır.

10.4.1. Timer2 Kesmesi (#INT_TIMER2)

TMR2 kaydedicisi içindeki değer 00h'tan başlayarak sayar ve PR2 kaydedicisi içine yazılan değere eşit olduğunda bir eşitlik sinyali üretilir. Daha önceden belirlenen postscale sayısı kadar eşitlik sinyali oluştuğunda Timer2 kesmesi oluşur. Timer2 kesmesine izin verme bit'i PIE1 kaydedicisinin 1.bit'i olan TMR2IE bit'idir. Kesme oluşma süresi ve kesme frekansı aşağıda verilen formül ile hesaplanır.

T_{Kesme} T_{osc}x(Timer2 bölme oranı)x(PR2 değeri+1)x(Postscaler değeri)

$$f_{Kesme} \quad \frac{1}{T_{Kesme}}$$

10.4.2. SETUP_TIMER_2() Fonksiyonu

Bu komut Timer2 zamanlayıcı ayarlarını yapmaya yarayan bir fonksiyondur.

setup_timer_2(*mod, periyod, postscale*);

Fonksiyondaki "*mod*" kısmına aşağıda belirtilen sabit tanımlamalardan biri yazılır. Bu fonksiyonda kullanılacak sabitler denetleyicinin başlık dosyasında (Örneğin, 16F877.h) yazılıdır.

MOD Kısmı Tanımlamları;
"T2_DISABLED" = Timer2'yi kapalı demek.
Timer2 Zamanlayıcı Bölme Oranları (Prescaler) Sabitleri;
"T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16"
Sabitin sonundaki rakamlar bölme oranını belirtir.

Periyod Kısmı;
Periyot kısmına 0 ile 255 arasında bir tamsayı girilir. Zamanlayıcı periyod değerine ulaşınca kendini sıfırlar. PR2 değerini temsil eder.
Postscale Kısmı;
Postscale kısmına 1 ile 16 arasında bir tamsayı girilir. Postscale değeri, Timer2 kesmesi oluşması için Timer2'nin kendisini kaç kez sıfırlaması gerektiğini bildirir.

Örnek;

```
setup_timer_2 ( T2_DIV_BY_16, 192, 3 );
```

komutu ile Timer2 biriminin bölme oranı 16, PR2 kaydedicisi değeri (periyot) 192 ve postscale değeri 3 olarak ayarlanıyor.

10.4.3. SET_TIMER2() Fonksiyonu

Bu komut Timer2 biriminin saymaya başlama değerini belirlemek için kullanılırlar. Tüm sayıcılarda olduğu gibi Timer2'de yukarı doğru sayar.

set_timer2(*değer*);

Fonksiyonda "*değer*" kısmına Timer2 biriminin saymaya başlama değeri yazılır. Bu değer TMR2 kaydedicisi içeriğidir. Timer2 birimi 8 bit'lik olduğu için buraya yazılacak değer 0 ile 255 arasında olmalıdır.
Örnek;

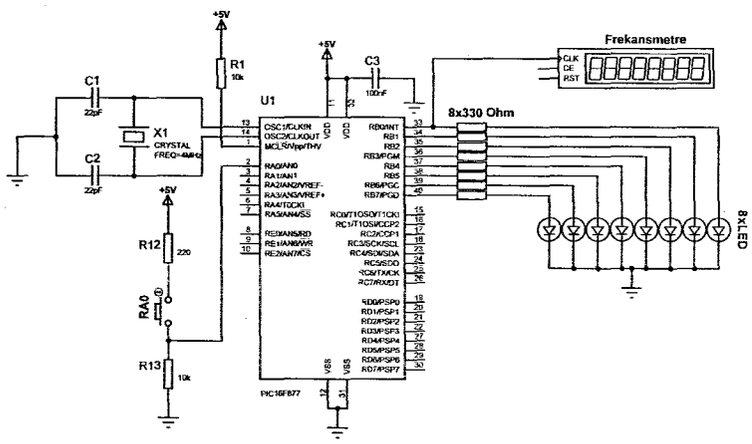
```
set_timer2(80);
```

komutu ile Timer2 birimi saymaya 80den itibaren başlar.

10.4.4. Timer2 Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada Timer2 birimi belli bir süre aralıklarla kesme sinyali verecek şekilde ayarlanmıştır. Kesme fonksiyonunda oluşan her kesme için "kesme" değişkeni içeriği 1 arttırılmaktadır. Ana fonksiyonda ise RA0 girişine bağlı butona basılıp basılmadığı test edilmektedir. Eğer butona basılmışsa o ana kadar oluşan kesme sayısı B port'una bağlı LED'lerde gösterilmektedir.



Şekil-10.11. *Timer2 uygulaması devre şeması.*

```
/*****************************************************************
      PIC16F877 ile Timer2 Uygulaması
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
```

```
#include <16f877.h>     // Kullanılacak denetleyicinin başlık dosyası
                        // tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

unsigned int kesme=0; // Tamsayı tipinde değişken tanımlanıyor

//****************** Timer2 Kesmesi ****************************
#int_timer2  // Timer2 kesmesi
void  timer2_kesme ()    // Kesme fonksiyonu ismi
{
      kesme++; // kesme değeri 1 arttırılıyor
       output_high(pin_b0); // RB0 lojik-1 yapılıyor
       output_low(pin_b0);  // RB0 lojik-0 yapılıyor
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x01);    // RA0 pin'i giriş
   set_tris_b(0x00);    // B port'u komple çıkış

   output_b(0x00);      // B port'u çıkışı ilk anda sıfırlanıyor

   setup_timer_2(T2_DIV_BY_16,250,16); // Timer2 ayarları yapılıyor

   enable_interrupts(INT_timer2);// int_timer2 kesmesini aktif yapar
   enable_interrupts(GLOBAL);    // Aktif edilen kesmelere izin ver

   while(1)     // Sonsuz döngü
   {
      if (input(pin_a0))    // RA0 butonuna basıldığında
         output_b(kesme);   // kesme değerini B port'una gönder
   }
 }
//*****************************************************************
```

Programda ana fonksiyonda, önceki tüm programlarda kullanılan `setup_timer_2(T2_DISABLED,0,1);` komutunun kullanılmadığına dikkat edin.

Önceki programlarda Timer2 birimi kullanılmadığından bu komut kullanılarak bu birim devre dışı bırakılmıştı. Şimdi ise bu uygulamada Timer2 birimi kullanıldığından bu komut kullanılmamıştır.

Ana fonksiyonda `setup_timer_2(T2_DIV_BY_16,250,16);` komutu ile Timer2 biriminin bölme oranının 16 olduğu, PR2 değerinin 250 olduğu ve postscale değerinin 16 olduğu belirtilmiştir. Ardından set_timer2(5); komutu ile de Timer2'nin saymaya başlama değeri 5 olarak verilmiştir.

`enable_interrupts(INT_timer2);` komutu ile Timer2 taşma kesmesi aktif edilmiş, `enable_interrupts(GLOBAL);` komutuyla da tüm aktif edilmiş kesmelere izin verilmiştir. Kesme fonksiyonu `#int_timer2` ve `void timer2_kesme ()` ifadeleri ile belirtilmiştir. Kesme fonksiyonunda her kesme oluştuğunda "kesme" değişkeninin değerinin 1 artması `kesme++;` komutu ile sağlanmıştır. Ayrıca kesme oluşumunun frekansını görmek maksadıyla RB0 pin'i her kesmede lojik-1'den lojik-0'a çekilmiştir. Böylece RB0 ucuna bir frekansmetre bağlanırsa kesme meydana gelme frekansı ölçülebilir.

Ana fonksiyondaki `while (1)` komutu ile sonsuz döngü oluşturulmuş ve bu döngüde `if (input(pin_a0))` komutu ile RA0 girişine bağlı butona basılıp basılmadığı kontrol edilmiştir. Eğer butona basılırsa `input()` komutu "1" değeri ile geri dönecek ve komut `if(1)` değerini alacaktır. Yani if komutu içindeki şart doğru olmuş olacaktır. Dolayısıyla `output_b(kesme);` komutu işlem görecektir. Bu komut ile de "kesme" değişkeni içeriği B port'una gönderilmiştir. Böylece butona basıldığında o ana kadar oluşa kesme sayısı LED'lerde görünecektir. "kesme" değişkeninin içeriğinin 255'e ulaştığında otomatik olarak sıfırlanacağı unutulmamalıdır (unsigned int komutu ile tanıtılan değişkenler maksimum 255 değerini alabilirler). Programda Timer2'nin kesme süresini hesaplarsak her 0,064256 sn'de bir kesme oluşmaktadır. Kesme süresi ve frekansı aşağıda hesaplanmıştır. RB0 ucuna frekasnmetre bağlanırsa hesaplanan değerin aynısı ölçü aletinde de görülecektir.

$$T_{Kesme} \quad T_{OSC}x(\text{Timer2 bölme oranı})x(\text{PR2 değeri}+1)x(\text{Postscaler değeri})$$

$$(1/f_{OSC})x16x(250 \quad 1)x16 \quad (1/1Mhz)x16x251x16 \quad 1 \ snx64256$$

$$64246 \ sn \quad 64,246msn \quad 0,064256sn$$

$$f_{Kesme} \quad \frac{1}{T_{Kesme}} \quad \frac{1}{0,06425sn} \quad 15,56Hz$$

10.5. WATCHDOG TIMER (WDT) BİRİMİ

WDT birimi önceden belirtildiği gibi denetleyici içinde gömülü bulunan ve ek olarak dışarıdan hiçbir elemanın bağlantısına gerek duymayan bir R/C osilatördür. Denetleyici osilatör girişlerine bağlanan (OSC1/CLKIN ve OSC2/CLKOUT)

osilatörden bağımsızdır. Normalde denetleycide osilatör bağlanmamış olsa veya denetleyici uyku (sleep) modunda olsa dahi WDT çalışmaya devam eder. WDT birimi 8 bit'lik bir zamanlayıcıdır. Bu nedenle WDT 0'dan 255'e kadar sayar ve 255'den sonra tekrar 0'a döndüğünde denetleyiciyi sıfırlar. WDT'ın taşma sinyali üretme süresi yaklaşık **18 ms**'dir. Bu değer ortam sıcaklığı, kullanılan denetleyici çeşidi ve Vdd değerlerine göre değişir. WDT taşma süresi arttırılmak isteniyorsa OPTION kaydedicinde bölme oranı değeri (prescaler) değiştirilir ve bu değerin WDT için geçerli olmasını sağlayan kaydedicinin 3. bit'i olan PSA bit'i "1" yapılır. Bu şekilde WDT taşma süresi en fazla yaklaşık **2,304 saniye** olabilir.

WDT normal çalışma esnasında belirlenen süre sonunda denetleyiciyi sıfırlar (resetler). Eğer denetleyici uyku (sleep) modunda iken WDT taşması meydana gelirse denetleyici uyku modundan çıkar ve programı kaldığı yerden icra etmeye devam eder.

WDT birimini açık (enable) veya kapalı (disable) yapmak için programlamada konfigürasyon bit'lerinde bunun belirtilmesi gerekir. Konfigürasyon bit'leri ile seçilen WDT birimi açık veya kapalı durumu yazılım aşamasında program komutlarıyla değiştirilemez. Yani WDT konfigürasyon bit'lerinde açık olarak belirtilmişse, denetleyiciye yüklenen program kodları WDT birimini kapatamaz, programla sadece WDT'ın içeriği sıfırlanabilir. CCS C'de PIC12, PIC14 ve PIC16 serisi denetleyiciler için WDT birimi açma ve kapatma işlemleri programın başında belirtilen şu komutlarla sağlanır.

```
#FUSE WDT      // WDT birimi açık
#FUSE NOWDT    // WDT birimi kapalı
```

CCS C'de PIC18 serisi denetleyiciler için WDT birimini açma ve kapatmak için ise şu komutlar kullanılır.

```
setup_wdt(WDT_ON);    // WDT birimi açık
setup_wdt(WDT_OFF);   // WDT birimi kapalı
```

Şekil-10.12'deki verilen WDT biriminin blok şemasında WDT çalışması daha net anlaşılabilir.

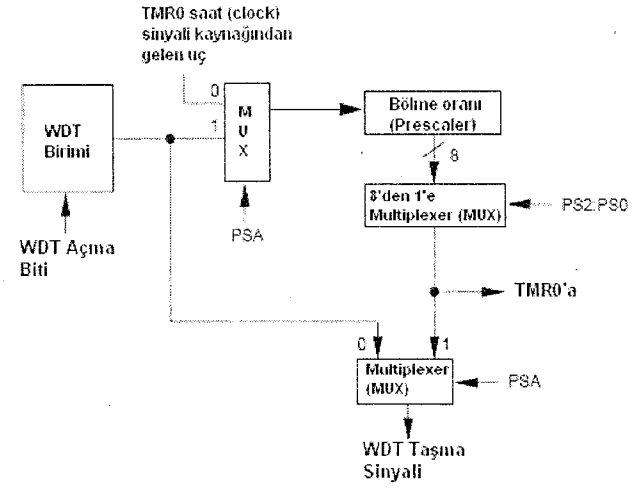
WDT'ın Türkçe karşılığı olarak **"bekçi köpeği"** tabiri kullanılmaktadır. Bunun nedeni WDT birimi, denetleyici içindeki programın veya mikrodenetleyicinin kontrol ettiği bir sistemin kararlı çalışmasının bekçiliğini yapmaktadır.

WDT biriminin kullanımı amacı olarak 2 neden söylenebilir. Bunlardan **ilki**, PIC denetleyicilerde programın istenmeyen bir şekilde kilitlenmesi durumunda PIC'in resetlenmesini sağlamak için kullanılır. Örneğin, WDT 2.3 saniyede bir taşma sağlayacak şekilde kurulur. Programda belli bir yere WDT'ı sıfırlama komutu yazılır. Bu komuttan önce programda diğer komutlar yer alır. Böylece program doğru çalıştığında hep, 2.3 saniye olmadan WDT'ı sıfırlamaktadır. Fakat program bir yerde kilitlenirse WDT'ı sıfırlama komutuna program geçemez. Böylece 2.3 saniye sonra

WDT taşması meydana gelir ve PIC resetlenir. Böylece kilitlenen programdan çıkılmış olunur.

İkinci kullanım amacı olarak da, tasarlanan bir mikrodenetleyicili sistemde meydana gelen bir arıza veya normal akış dışı bir olay meydana geldiğinde denetleyicinin belirlenen süre sonunda kendisini resetlemesidir. Bir örnekle olayı açıklayalım. Bir araba fabrikasında bir kaynak robotunun 1.5 saniyede bir gelen gelen parçaya lazer ile kaynak yaptığını düşünün. Programla her parça gelişinde WDT taşmasının sıfırlanması sağlanmış olsun. Kaynak yapılacak parçanın geliş bandı hattında bir sorun oluştuğunu ve parçanın kaynak robotuna ulaşamadığını düşünün. Bu durumda WDT, parça gelmediği için sıfırlanmayacak ve belli bir gecikmeden sonra WDT taşması meydana gelecektir. Bu durumda PIC resetlenecektir ya da bir uyarı sistemi tetiklenmesi sağlanabilecektir.

WDT birimi kullanıldığında istenmeyen zamanlarda PIC'in resetlenmesini önlemek için programda gerekli yerlerde mutlaka WDT'ı sıfırlayan komut kullanılmalıdır.



Şekil-10.12. *WDT birimi blok şeması.*

10.5.1. SETUP_WDT() Fonksiyonu

Watchdog Timer ayarları için kullanılan bir fonksiyondur.

setup_wdt (*mod*);

Fonksiyonda parantez içine WDT taşma süresini belirleyen sabit tanımlamalar yazılır. Bu fonksiyonda *"mod"* kısmına şu bölme sabitleri (prescaler) yazılabilir. **Bu bölme değerleri (WDT taşma süresini belirleyen sabitler) 12 bit ve 14 bit PIC denetleyiciler için geçerlidir.**

12 bit ve 14 bit PIC denetleyiciler için kullanılan sabitler;

WDT_18MS = WDT taşma süresi 18 ms.
WDT_36MS = WDT taşma süresi 36 ms.
WDT_72MS = WDT taşma süresi 72 ms.
WDT_144MS = WDT taşma süresi 144 ms.
WDT_288MS = WDT taşma süresi 288 ms.
WDT_576MS = WDT taşma süresi 576 ms.
WDT_1152MS = WDT taşma süresi 1152 ms (1.152 saniye).
WDT_2304MS = WDT taşma süresi 2304 ms (2.304 saniye).

16 bit PIC18FXXX için şu sabitler kullanılır;

WDT_ON = WDT aktif, WDT_OFF = WDT pasif.

Peki 16 bit'lik PIC denetleyicilerde (18FXXX serisi) WDT taşma süresi nasıl belirlenecek. Bu iş için 18FXXX serisi denetleyicilerde #fuse komutu kullanılır.

#fuse WDTx

komuttaki x yerine 1,2,3 gibi sayı değerleri yazılır. Şu şekilde de WDT süresi hesaplanır. 4ms*x değeri WDT taşma süresini verir.

```
#fuse WDT1      // 4msx1=4 msn WDT taşma süresi.
```

10.5.2. RESTART_WDT() Fonksiyonu

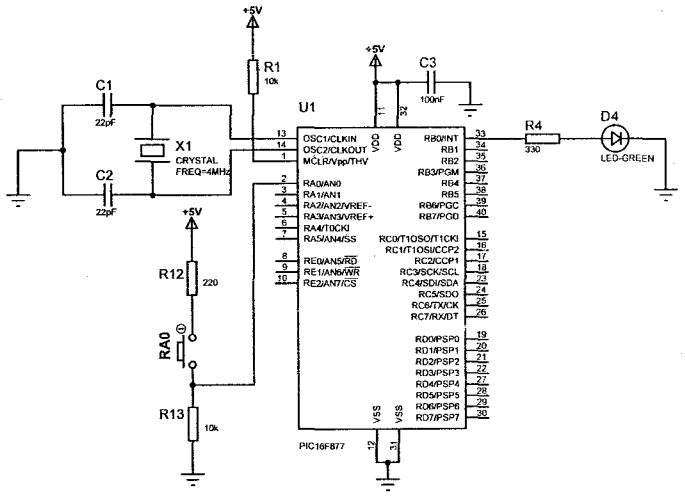
WDT'ı sıfırlamak (resetlemek) için kullanılan bir fonksiyondur.

restart_wdt ( );

10.5.3. WDT Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada WDT taşma süresi 2.304 saniyeye ayarlanmıştır. Devrede ilk anda RB0 pin'ine bağlı LED yakılmış ardından da PIC uyku (sleep) moduna alınmıştır. Her 2.304 saniye WDT taşması meydana gelmekte ve PIC uyku modundan çıkmaktadır. Böylece LED ilk konumunun tersi duruma geçirilmektedir yani söndürülmektedir. RA0 ucuna bağlanan buton ile de WDT'ın sıfırlanması sağlanmıştır.



Şekil-10.13. *WDT uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile WDT Uygulaması
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>   // Kullanılacak denetleyicinin başlık dosyası
                      // tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,WDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 birimi devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x01);    // RA0 pin'i giriş
   set_tris_b(0x00);    // B port'u komple çıkış

   output_b(0x00);      // B port'u çıkışı ilk anda sıfırlanıyor
```

```
   setup_WDT(WDT_2304MS); // WDT ayarları yapılıyor

   while(1)     // Sonsuz döngü
   {
      output_toggle(pin_b0); // RB0 ucunun çıkış durumu tersleniyor

      while (input(pin_a0))//Eğer RA0 girişindeki butona basıldı ise
         restart_wdt();      // WDT'ı sıfırla

      sleep();    // Denetleyici uyku moduna alınıyor.
   }
 }
//*****************************************************************
```

Programda ilk dikkat edilmesi gereken konfigürasyon bit'leri komutu belirtilirken her zaman kullanılan `#fuses XT,NOWDT,...` kısmı bu sefer `#fuses XT,WDT,...` olarak kullanılmıştır. Çünkü WDT birimini bu programda kullanacağız. Ana fonksiyonda `setup_WDT(WDT_2304MS);` komutu ile WDT kesme taşma süresi 2.304 saniyeye ayarlanmıştır. `while(1)` komutu ile belirlenen sonsuz döngüde, `output_toggle(pin_b0);` komutu ile RB0 ucunun çıkış durumu terslenmektedir. Böylece programda bu komuta her gelindiğinde LED durum değiştirmektedir. Sonsuz döngü içindeki diğer `while (input(pin_a0))` döngü komutu ile RA0 girişine bağlı butona basılıp basılmadığı kontrol edilmektedir. **Bu butona basıldığı sürece WDT sıfırlanmaktadır. Böylece LED'in konumu buton basılı kaldığı sürece sabit kalmaktadır.** Eğer buton basılı değilse programda `sleep();` komutu ile PIC uyku moduna alınmaktadır. PIC uyku modunda iken her 2.304 saniyede WDT taşması olacağından her 2.304 saniyede PIC resetlenecek ve her resetlemede LED konum değiştirerek bir yanıp bir sönecektir.

11. Bölüm

CAPTURE/COMPARE/PWM BİRİMİ İŞLEMLERİ

- CCP Birimine Genel Bakış
- Capture Modu
- Compare Modu
- PWM Modu
- SETUP_CCPx () Fonksiyonu
- SET_PWMx_DUTY () Fonksiyonu
- Capture Uygulaması
- Compare Uygulaması
- PWM Uygulaması

11.1. CCP BİRİMİNE GENEL BAKIŞ

CCP birimi içinde 3 değişik modda çalışan yani 3 değişik modül içeren bir birimdir. Bu çalışma modları, Capture (yakalama), Compare (karşılaştırma) ve PWM (Pals Genişlik Modülasyonu) modudur. PIC16F877 içinde 2 adet CCP birimi vardır. Bunlar CCP1 ve CCP2'dir. Her CCP birimi, 16 bit'lik capture kaydedicisi olarak, 16 bit'lik compare kaydedicisi veya 10 bit'lik PWM Master/Slave görev çevrimi kaydedicisi olarak çalışan 16 bit'lik kaydediciye sahiptir. CCP birimi donanımı sayesinde program komutlarını kullanmadan yakalama (capture), karşılaştırma (compare) ve PWM sinyali üretme işlemleri kolayca yapılabilir. Bu işlemler için her CCP birimi çalışma modu denetleyici içinde bulunan bir zamanlayıcı/sayıcı birimini kaynak olarak kullanır. Tablo-11.1'de bu CCP biriminin kullandığı zamanlayıcı/sayıcı kaynakları verilmiştir.

CCP Modu	Kullandığı Zamanlayıcı
Capture	Timer1
Compare	Timer1
PWM	Timer2

Tablo-11.1. *CCP biriminin kullandığı zamanlayıcı/sayıcı birimleri.*

CCPx birimini **CCP1CON** ve **CCP2CON** kaydedicileri kontrol etmektedir. İki kaydedicinin yapısı aynıdır. Şekil-11.1'de CCPxCON kaydedici bit'leri ve bit'lerin görevleri verilmiştir. Şekilde bit isimlerindeki x yerine CCP1CON kaydedicisi için 1, CCP2CON kaydedicisi için 2 numarası gelmektedir.

CCPxCON KAYDEDİCİSİ

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

Şekil-11.1. *CCPxCON kaydedicisi.*

Bit 7-6 = Kullanılmayan pin'ler. Lojik-0 olarak okunur.
Bit 5-4 **CCPxX-CCPxY** = PWM düşük değerlikli bit'leri.
(Bu bit'ler Capture ve Compare modlarında kullanılmaz.)

PWM görev çevriminin düşük değerlikli 2 bit'inin değerini belirtir. 10 bit'lik görev çevrimi değerinin yüksek değerlikli 8 bit'lik kısmı ise CCPRxL kaydedicisinde bulunur.

Bit 3-0 **CCPxM3:CCPxM0** = CCPx modu seçme bit'leri
0000 = Capture/Compare/PWM Kapalı.
0100 = Capture mod, her düşen kenarda.
0101 = Capture mod, her yükselen kenarda.
0110 = Capture mod, her dördüncü yükselen kenarda.
0111 = Capture mod, her onaltıncı yükselen kenarda.
1000 = Karşılaştırma modunda Eşleşme olayı meydana geldiğinde CCPx pin'i "1" olsun ve CCPx kesmesi çağrılsın.
1001 = Karşılaştırma modunda Eşleşme olayı meydana geldiğinde CCPx pin'i "0" olsun ve CCPx kesmesi çağrılsın.
1010 = Karşılaştırma modunda Eşleşme olayı meydana geldiğinde CCPx pin'i durumu değişmesin ve CCPx kesmesi çağrılsın.
1011 = Karşılaştırma modunda Eşleşme olayı meydana geldiğinde CCPx pin'inin durumu değişmesin ve Timer1 resetlensin.
11xx = PWM modu.

11.2. CAPTURE MODU

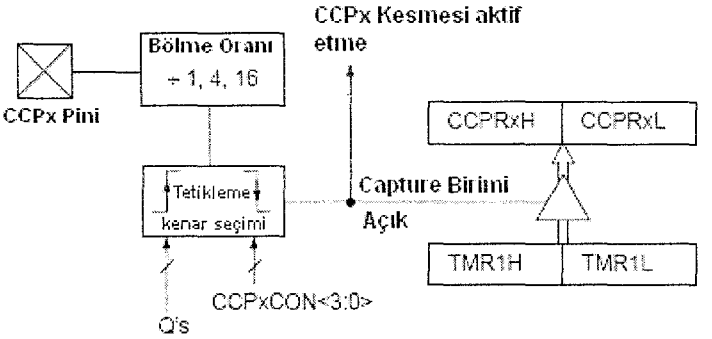
CAPTURE modunda belirlenen RCx/CCPx (CCP1 birimi için RC1/CCP1 pin'i, CCP2 birimi için RC2/CCP2 pin'i) pin'inden gelen clock sinyali aşağıda verilen şartlara göre değerlendirilir. Bu şartlardan biri sağlandığında (yakalama olayı meydana geldiğinde) TMR1 kaydedicisi içeriği CCPRxH:CCPRxL kaydedicisine aktarılır (Kaydedilecek değer 16 bit'lik olduğundan iki adet 8 bit'lik kaydedicide saklanır) ve bir kesme oluşur. İkinci bir yakalama olayı meydana geldiğinde yine o anki TMR1 değeri CCPRxH:CCPRxL kaydedicisine aktarılır, eski CCPRxH:CCPRxL değeri silinmiş olur.

Sinyalin her düşen kenarında
Sinyalin her yükselen kenarında
Sinyalin her 4. yükselen kenarında
Sinyalin her 16. yükselen kenarında

CCPRxH kaydedicisi 16 bit'lik değerin yüksek değerlikli 8 bit'ini, CCPRxL kaydedicisi ise 16 bit'lik değerin düşük değerlikli 8 bit'ini içerir. CCPx pin'inden sinyal alımı için bu ucun giriş olarak yönlendirilmesi gerekmektedir. Tanımlamalarda yer alan "x" yerine kullanılan CCP biriminin numarası yazılır.

Not: CCS C'de CCPx birimi ile işlem yaparken esneklik sağlaması açısından denetleyici başlık dosyasında CCPR1 kaydedicisi CCP_1 (16 bit'lik), CCPR2 kaydedicisi CCP_2 (16 bit'lik) olarak tanımlanmıştır. Ayrıca 16 bit'lik CCPR1 kaydedicisinin 8 bit'lik yüksek değerlikli kısmını tutan CCPR1H kaydedicisi CCP_1_HIGH, düşük değerlikli 8 bit'ini tutan CCPR1L kaydedicisi CCP_1_LOW olarak tanımlanmıştır. Aynı şekilde CCPR2H kaydedicisi CCP_2_HIGH, CCPR2L kaydedicisi CCP_2_LOW olarak tanımlanmıştır. Bu tanımlamaları kullanarak bu kaydedicilere istenen değerler direkt olarak yazılabilir. Aşağıda bazı örnekler verilmiştir.

```
CCP_1=600;  // CCPR1 kaydedicisi içeriğine 600 değeri yüklenmiştir.
CCP_1_HIGH=0x45;  // CCPR1H kaydedicisi 8 bit 45h değeri yüklenmiştir.
```

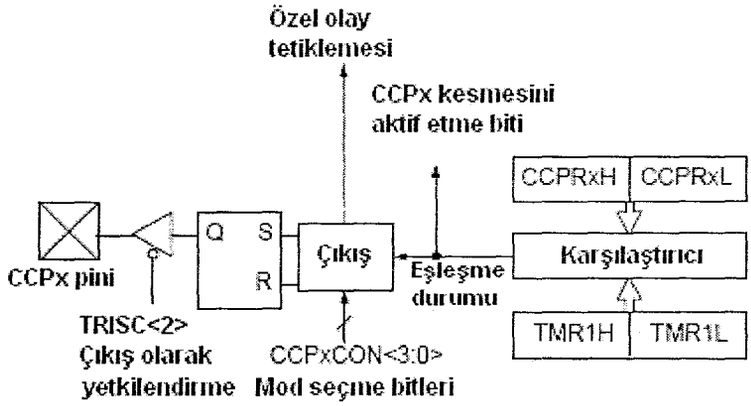


Şekil-11.2. *CAPTURE modu blok şeması.*

11.3. COMPARE MODU

CCP biriminin COMPARE modunun çalışma prensibi şöyledir. CCPRx kaydedicisinin içindeki değer ile Timer1 kaydedicisi olan TMR1'in içindeki değer sürekli olarak karşılaştırılır. Karşılaştırma işlemi sonucunda eşleşme meydana gelirse CCPx kesmesi meydana gelir ve CCPxCON kaydedicisinde belirtilen bit ayarları ile aşağıda belirtilen 4 olaydan istenen biri gerçekleştirilebilir. Şekil-11.3'de COMPARE modunun blok diyagramı görülmektedir. CCPx ifadesindeki x yerine hangi CCP modülü kullanılıyorsa onun numarası yazılır. Örneğin CCP2 modülü için x yerine 2 yazılmalıdır.

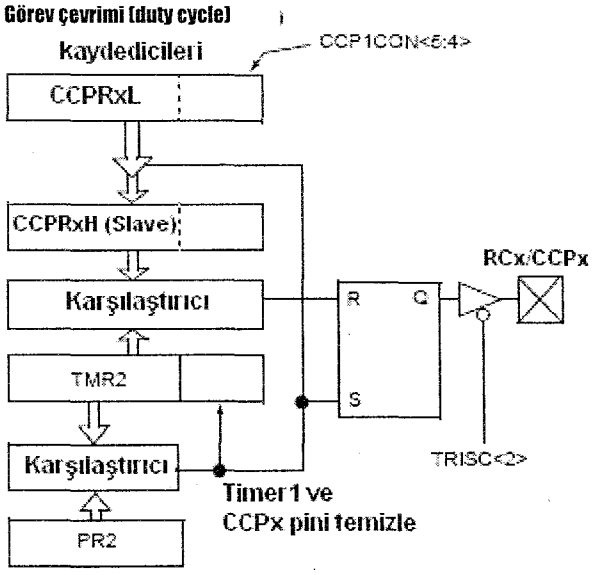
CCPx pin'i lojik-1 olsun, kesme meydana gelsin
CCPx pin'i lojik-0 olsun, kesme meydana gelsin
CCPx pin'inin durumu değişmesin, kesme meydana gelsin
CCPx pin'inin durumu değişmesin, kesme meydana gelsin ve Timer1 resetlensin.



Şekil-11.3. *COMPARE modu blok diyagramı.*

11.4. PWM MODU

CCP biriminin PWM modu, istenen CCPx ucundan istenen görev çevrimine (duty cycle - doluluk oranı) sahip PWM sinyali elde etmek için kullanılır. PWM birimi Timer2 zamanlayıcısını kullanır. PIC16F877'de iki adet CCP modülü olduğundan 2 adet de PWM çıkış ucu vardır. PWM modunda, PWM çıkışı olarak kullanılan RC1/T1OSI/CCP2 ve RC2/CCP1 pin'leri çıkış olarak yönlendirilmelidir. Aşağıda Şekil-11.4'de PWM modunun çalışmasının blok diyagramı verilmiştir.



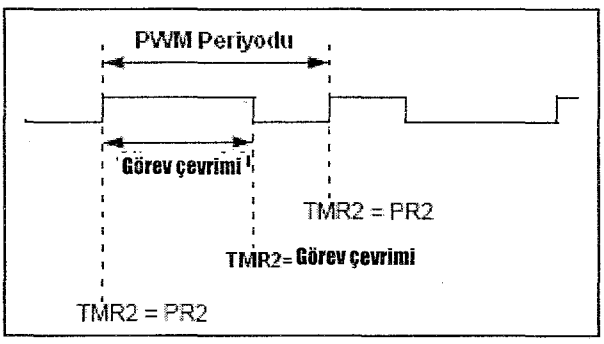
Şekil-11.4. *PWM modu blok diyagramı.*

Blok şemada da görüldüğü gibi oluşturulacak PWM sinyali denetleyici osilatör frekansına, PR2 değerine ve Timer2 zamanlayıcı/sayıcı biriminin bölme oranı değeri ile belirlenir. Aşağıda oluşturulan PWM sinyalinin, periyod ve frekans hesaplanması için kullanılacak formül ile oluşturulan bir PWM dalga şekli verilmiştir.

<u>PWM sinyalinin periyodu ve frekansı;</u>

$$T_{PWM}=T_{OSC}.4.(PR2+1).(\text{TMR2 bölme oranı})=\frac{1}{f_{OSC}}.4.(PR2+1).(\text{TMR2 bölme oranı})$$

$$f_{PWM}=\frac{1}{T_{PWM}}$$



Şekil-11.5. *Oluşturulan PWM dalga şekli.*

PWM dalga şeklinde görüldüğü gibi PWM sinyalin bir görev çevrimi -doluluk oranı- (duty cycle) bölümü vardır. Bu bölüm PWM sinyalinde lojik-1 (High) olan süredir. Bu görev çevrimi süresi PWM periyodundan uzun olamaz. PWM modülü 10 bit'lik bir çözünürlüğe sahip PWM sinyali üretebilir. Bu 10 bit'lik değerin 8 bit'i kısmı CCPRxL kaydedicisinde, düşük değerlikli 2 bit'i ise CCPxCON kaydedicisinin 5 ve 4. bit'leri ile ifade edilir. Böylece 10 bit'lik değer tanımlanmış olur. PWM sinyalinin görev çevrimi periyodu (süresi) aşağıdaki formül ile hesaplanır. Formüldeki istenen değerler program vasıtası ile değiştirilerek PWM sinyalinin görev çevrimi süresi değiştirilmiş olur.

PWM sinyalinin görev çevrimi (duty cycle) periyodu:

T_{PWM_DUTY}=(CCPR1L:CCP1CON<5:4>).T_{OSC}.(TMR2 bölme oranı)

PWM sinyalinin görev çevrimi süresi PWM sinyalinin periyodundan daha büyük olmaması gereklidir. Programlamada bu hususu göz önüne alarak görev çevrimi süresi değeri tespit edilmelidir.

PWM modülünde oluşturulan PWM sinyalinin bir de **çözünürlük** değeri vardır. **PWM çözünürlüğü**, PWM sinyalinin kaç adet eşit parçada değiştirilebileceğini belirtir. PWM çözünürlüğü aşağıda verilen formül ile hesaplanır.

$$\text{PWM Çözünürlüğü}=\frac{\log \frac{F_{OSC}}{F_{PWM}}}{\log 2}\,bit$$

PWM çözünürlüğü en 2, en fazla 10 bit olabilir.

11.5. SETUP_CCPX() FONKSİYONU

Bu fonksiyon ile PIC donanımı içinde bulunan (her modelde yoktur) CCP modülü ile ilgili ayarlamalar yapılır.

setup_ccp1(*mod*); , setup_ccp2(*mod*); , setup_ccp3(*mod*); ...

CCP modülü ister Capture, ister Compare, isterse PWM modunda çalıştırılır. Fonksiyondaki *"mod"* kısmına aşağıda verilen sabit tanımlamalar yazılır. Bu tanımlamalar yazılırken hangi mod (Capture, Compare, PWM) kullanılacaksa o moda uygun tanımlamalar kullanılır. Fonksiyondaki x kısmına kullanılan CCP modülü birimi numarası yazılır.

CCP birimini devre dışı yapmak için:

CCP_OFF = CCP birimi devre dışı

Capture Mode;

CCP_CAPTURE_FE = Düşen kenarları (falling edge) yakala
CCP_CAPTURE_RE = Yükselen kenarları (rising edge) yakala
CCP_CAPTURE_DIV_4 = Her dördüncü palsten sonra yakala
CCP_CAPTURE_DIV_16 = Her onaltıncı palsten sonra yakala

Compare Mode;

CCP_COMPARE_SET_ON_MATCH = Eşleşme olayı meydana geldiğinde CCPx pin'i "1" olsun, kesme meydana gelsin.
CCP_COMPARE_CLR_ON_MATCH = Eşleşme olayı meydana geldiğinde CCPx pin'i "0" olsun, kesme meydana gelsin.
CCP_COMPARE_INT = Eşleşme olayı meydana geldiğinde sadece kesme meydana gelsin, CCPx değişmesin.
CCP_COMPARE_RESET_TIMER = Eşleşme olayı meydana geldiğinde kesme meydana gelsin, CCPx pin'ini durumu değişmesin ve Timer1 sıfırlansın.

Not: CCPx ifadesindeki x yerine hangi CCP modülü kullanılıyorsa onun numarası yazılır. Örneğin CCP2 modülü için x yerine 2 yazılmalıdır.

PWM Mode;

CCP_PWM = PWM modunu aç.

11.6. SET_PWMX_DUTY() FONKSİYONU

PWM modunda oluşturulan PWM sinyalinin görev çevrimi süresini belirlemeye yarayan bir fonksiyondur. Fonksiyonda *"değer"* kısmına 10 bit'lik sabit veya değişken yazılabilir. Bu *"deger"* kısmı PWM görev çevrimi hesaplama formülünde bulunan (CCPR1L:CCP1CON<5:4>) kısmın karşılığıdır. *"değer"* kısmına 8 bit'lik bir değer girildiğinde derleyici yazılan bu değeri 0 kullanarak kaydırır ve 10 bit'lik değere çevirir. Fonksiyondaki x kısmına kullanılan CCP modülü birimi numarası yazılır.

set_pwm1_duty(*değer*);, set_pwm2_duty(*değer*);, set_pwm3_duty(*değer*); ...

```
set_pwm1_duty(500); // PWM görev çevrimi süresi 500 olarak ayarlanıyor.
```

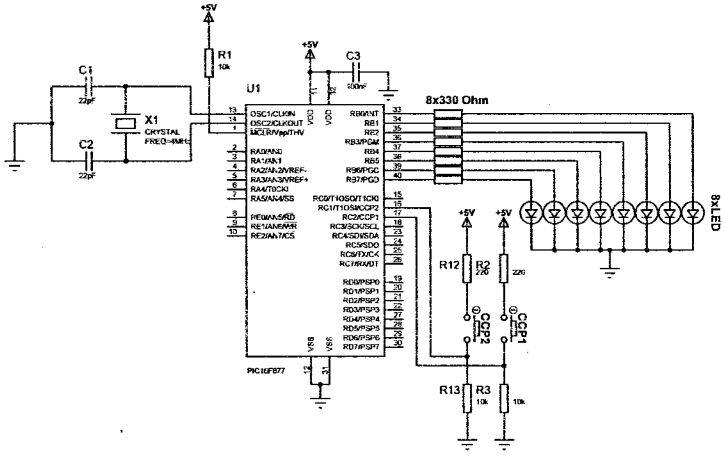
11.7. CAPTURE MODU UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Capture modu uygulamamızda PIC16F877 içinde bulunan 2 adet CCP1 ve CCP2 biriminin ikisininde CAPTURE (yakalama) modu kullanılmıştır. Uygulamada B port'una 8 adet LED, RC2/CCP1 ve RC1/CCP2 pin'lerine de buton bağlanmıştır. Butonlar vasıtası ile CAPTURE (yakalama) olayının oluşması için sinyal girişi sağlanmıştır. Programda her CCP1 yakalama kesmesi meydana geldiğinde "i" değişkeninin bir artması sağlanmış ve bu "i" değeri B port'una gönderilerek LED'lerde görünmesi sağlanmıştır. Her CCP2 yakalama kesmesi meydana geldiğinde ise "i" değerinin bir azalması sağlanmış ve elde edilen i değeri B port'una gönderilerek sonucun LED'lerde görünmesi sağlanmıştır.

- Devrenizi PIC PROG/DEKA 16F87x deneme kartında denemek için küçük bir işlem yapmanız gereklidir. Çünkü deneme kartında C port'u başka amaçlar için başka eleman giriş ve çıkışlarına bağlanmıştır. Bu nedenle RC1 ve RC2 pin'lerine bağlı butonlar yerine RA0 ve RA1 butonları kullanılmalıdır. Deneme kartının sol tarafında yer alan Port A I/O çıkış pin'lerinden RA0 pin'ine zil teli kablosunun bir ucu yerleştirilmeli, kablonun diğer ucu ise PIC16F877 denetleyicisinin takıldığı sokette RC2 pin'ine sıkıştırılmalıdır. Aynı mantıkla RA1 pin'i ile RC1 ucu zil teli kablosu ile bağlanmalıdır. Bu sayede devrenizi deneyebilirsiniz. Fakat RC1 ucu IR alıcı elemanına bağlandığı için RC1 ucuna bağlı butona bir basmadan sonra ikinci basma için belli bir süre ara vermeniz gerekmektedir.

- Eğer deneme kartınızı zil teli ile yaptığınız ekstra bağlantılardan dolayı programlama da sorun yaşarsanız. Deneme kartınızın harici enerjisini kestikten sonra programlama işleminizi yapın ve daha sonra tekrar deneme kartınıza enerji vererek programınızın çalışmasını gözleyiniz.



Şekil-11.6. *CAPTURE modu uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Capture Modu Uygulaması
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.

// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

int i=0; // Tamsayı tipinde değişken tanımlanıyor

//*********** CCP1 KESMESİ *****************
#INT_CCP1   // INT_CCP1 kesmesi fonksiyonu
void yakala1_kesmesi ()
{
   i++;              // i değerini 1 arttır
   output_b(i);      // i değerini B port'una gönder
}

//*********** CCP2 KESMESİ *****************
#INT_CCP2   // INT_CCP2 kesmesi fonksiyonu
void yakala2_kesmesi ()
{
   if(i==0)          // Eğer i değeri sıfırsa
      i=1;           // i değeri içeriği 1 olsun

   i--;              // i değerini 1 azalt
   output_b(i);      // i değerini B port'una gönder
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı

   set_tris_b(0x00);   // B port'u komple çıkış
   set_tris_c(0x07);   // RC0, RC1 ve RC2 pin'i giriş

   output_b(0x00); // İlk anda B port'u çıkışı sıfırlanıyor
```

```
   enable_interrupts(INT_CCP1);  // INT_CCP1 kesmesi aktif yapılıyor
   enable_interrupts(INT_CCP2);  // INT_CCP2 kesmesi aktif yapılıyor
   enable_interrupts(GLOBAL);    // Aktif edilen kesmelere izin ver

   setup_ccp1(CCP_CAPTURE_DIV_4); // CCP1 kesmesi her 4 yükselen
                                  // kenarda 1 yakalama yapacak
   setup_ccp2(CCP_CAPTURE_RE);    // CCP2 kesmesi her yükselen
                                  // kenarda 1 yakalama yapacak

   CCP_1_HIGH=0x00; // CCPR1H kaydedicisi sıfırlanıyor
   CCP_1_LOW=0x00;  // CCPR1H kaydedicisi sıfırlanıyor

   CCP_2_HIGH=0x00; // CCPR2H kaydedicisi sıfırlanıyor
   CCP_2_LOW=0x00;  // CCPR2H kaydedicisi sıfırlanıyor

   while(1); // Sonsuz döngü
}
//*****************************************************************
```

Programda ilk başta dikkat edilmesi gereken nokta diğer programlarımızda genelde hep devre dışı bıraktığımız Timer1, CCP1 ve CCP2 birimleri bu sefer devre dışı bırakılmamıştır. Çünkü bu programımızda bu birimleri kullanacağız. Programda port giriş/çıkış yönlendirmeleri `set_tris_b(0x00);` ve `set_tris_c(0x07);` komutları ile yapıldıktan sonra, `output_b(0x00);` komutu ile ilk anda B port'u çıkışına bağlı LED'lerin sönük olması sağlanmıştır. Programda daha sonra `enable_interrupts(INT_CCP1);` ve `enable_interrupts(INT_CCP2);` komutları ile CCP1 ve CCP2 birimi kesmeleri aktif yapılmış, ardından `enable_interrupts(GLOBAL);` komutu ile de tüm aktif edilen kesmelere izin verilmiştir. `setup_ccp1(CCP_CAPTURE_DIV_4);` komutu ile CCP1 biriminin CAPTURE modunun kullanılacağı ve RC2 pin'inden gelen her 4 yükselen kenarda bir CCP1 kesmesi meydana geleceği belirtilmiştir. Aynı mantıkla `setup_ccp2(CCP_CAPTURE_RE);` komutu ile de CCP2 biriminin CAPTURE modunun kullanılacağı ve RC1 pin'inden gelen her yükselen kenarda bir CCP2 kesmesi meydana geleceği belirtilmiştir. Programdaki `CCP_1_HIGH=0x00;... CCP_2_LOW=0x00;` komutları ile CCPR1H:CCPR1L ve CCPR2H:CCPR2L kaydedicileri içerikleri sıfırlanmıştır.

CCS C'de; 16 bit'lik CCPR1 kaydedicisinin 8 bit'lik yüksek değerlikli kısmını tutan CCPR1H kaydedicisi CCP_1_HIGH, düşük değerlikli 8 bit'ini tutan CCPR1L kaydedicisi CCP_1_LOW olarak tanımlanmıştır. Aynı şekilde CCS C'de; CCPR2H kaydedicisi CCP_2_HIGH, CCPR2L kaydedicisi CCP_2_LOW olarak tanımlandığını hatırlayınız.

CCP1 kesme fonksiyonunda `i++;` ve `output_b(i);` komutları ile her CCP1 kesmesi meydana geldiğinde i değişkeni değerinin 1 arttırılması ve i değerinin B port'una bağlı LED'lerde gösterilmesi sağlanmıştır. CCP2 kesme fonksiyonunda ise `if(i==0)` komutu ile i değerinin sıfıra eşit olup olmadığı kontrol edilmiş ve eğer i değeri sıfıra eşitse `i=1;` komutu ile i değeri 1'e eşitlenmiştir. Fonksiyonda `i--;`

ve `output_b(i);` komutları ile her CCP2 kesmesinde i değerinin 1 azaltılması ve bu değerin B port'una bağlı LED'lerde görünmesi sağlanmıştır. Eğer i değişkeni 0'a eşit ise i değeri if kontrol fonksiyonunda 1 yapılarak sonuçta i değerinin en küçük değerinin hep 0 olması sağlanmıştır.

Önceden anlatıldığı gibi, belirlenen şarta göre bir yakalama (Capture) olayı meydana geldiğinde TMR1 değeri CCPRxH:CCPRxL kaydedicisine aktarılır. Bu özellik ve Timer1 kesmesi bu uygulamada kullanılmamıştır. İstenirse bu özelliklerden de faydalanarak değişik amaçları karşılayan uygulamalar yapılabilir.

11.8. COMPARE MODU UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu örnekte de, CCP biriminin COMPARE (karşılaştıma) modu özelliği incelenmiştir. Bilindiği gibi COMPARE (karşılaştırma) modu çalışması şu şekildeydi. CCPRx kaydedicisinin içindeki değer ile Timer1 kaydedicisi olan TMR1'in içindeki değer sürekli olarak karşılaştırılmakta, karşılaştırma işlemi sonucunda eşleşme meydana geldiğinde CCPx kesmesi meydana gelmekte ve CCPxCON kaydedicisinde belirtilen bit ayarları ile 4 olaydan istenen biri gerçekleştirilebilinmekteydi. Bu uygulamada Timer1, harici senkronize saat sinyali girişi ile artacak şekilde ayarlanmıştır. Timer1 harici saat sinyali girişi olan RC0/T1OSO/T1CKI pin'ine harici sinyal için buton bağlanmıştır. Böylece TMR1 kaydedicisi değerinin buton ile arttırılabilmesine olanak verilmiştir. COMPARE olayı meydana geldiğinde gerçekleştirilecek iş olarak ta 4 olaydan aşağıdaki olayın meydana gelmesi sağlanmıştır.

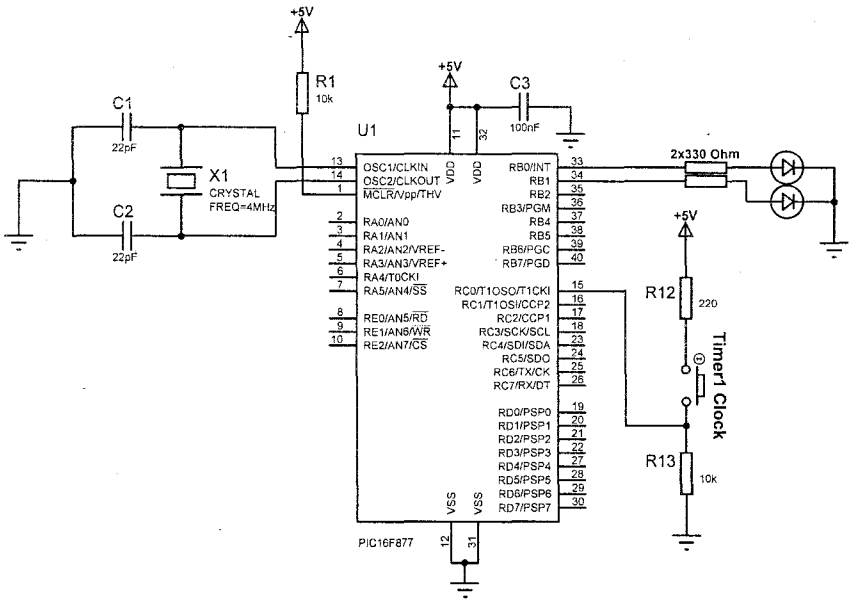
CCPx pin'inin durumu değişmesin, kesme meydana gelsin

Programla CCPR1H:CCPR1L kaydedicisine 5, CCPR2H:CCPR2L kaydedicisine 10 değeri yüklenmiştir. Böylece buton girişinden 5 tetikleme sinyali geldiğinde CCP1 biriminde karşılaştırma sonucunda eşitlik olacak ve CCP1 kesmesi meydana gelecektir. Buton girişinden 10 tetikleme sinyali geldiğinde CCP2 biriminde karşılaştırma sonucunda eşitlik olacak ve CCP2 kesmesi meydana gelecektir. Her CCP1 kesmesinde RB0 pin'ine bağlı LED'in durum değiştirmesi, her CCP2 kesmesinde ise RB1 pin'ine bağlı LED'in durum değiştirmesi sağlanmıştır.

- Uygulama devresini Proteus programınızda denerken butona 5 defa veya 10 defa basma sayısından daha az bir sayıda basınca kesme meydana gelebilir. Çalışmanın tam olarak benzetimini yapmak için buton yerine "Logic State" elemanını bağlayınız. "1" ve "0" değerlerini "Logic State" elemanından vermeniz halinde sonucu tam olarak görebilirsiniz.

- Uygulama devresini PIC PROG/DEKA 16F87x deneme kartında denemek için küçük bir işlem yapmanız gereklidir. Deneme kartının sol tarafında yer alan Port A I/O çıkış pin'lerinden RA0 pin'ine zil teli kablosunun bir ucu yerleştirilmeli, kablonun

diğer ucu ise PIC16F877 denetleyicisinin takıldığı sokette RC0 pin'ine sıkıştırılmalıdır.



Şekil-11.7. *Compare uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Compare Modu Uygulaması
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

//*********** CCP1 KESMESİ *****************
#INT_CCP1   // INT_CCP1 kesmesi fonksiyonu
void karsilastir1_kesmesi ()
{
   output_toggle(pin_b0);  // RB0 çıkışı tersleniyor
}

//*********** CCP2 KESMESİ *****************
#INT_CCP2   // INT_CCP2 kesmesi fonksiyonu
```

```
void karsilastir2_kesmesi ()
{
   set_timer1(0);          // TMR1 içeriği sıfırlanıyor
   output_toggle(pin_b1);  // RB1 çıkışı tersleniyor
}
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı

   set_tris_b(0x00);   // B port'u komple çıkış
   set_tris_c(0x01);   // RC0 pin'i giriş

   output_b(0x00); // İlk anda B port'u çıkışı sıfırlanıyor

   enable_interrupts(INT_CCP1);  // INT_CCP1 kesmesi aktif yapılıyor
   enable_interrupts(INT_CCP2);  // INT_CCP2 kesmesi aktif yapılıyor
   enable_interrupts(GLOBAL);    // Aktif edilen kesmelere izin ver

   setup_ccp1(CCP_COMPARE_INT);  // CCP1 birimi COMPARE modunda
   setup_ccp2(CCP_COMPARE_INT);  // CCP2 birimi COMPARE modunda

   setup_timer_1(T1_EXTERNAL_SYNC | T1_DIV_BY_1); // Timer1 ayarları
                                                  // yapılıyor

   CCP_1_HIGH=0x00; // CCPR1H kaydedicisi sıfırlanıyor
   CCP_1_LOW=0x05;  // CCPR1L kaydedicisine 05h değeri yükleniyor.

   CCP_2_HIGH=0x00; // CCPR2H kaydedicisi sıfırlanıyor
   CCP_2_LOW=0x0A;  // CCPR2L kaydedicisine 0Ah değeri yükleniyor.

   set_timer1(0);   // TMR1 içeriği sıfırlanıyor

   while(1);        // Sonsuz döngü
}
//*****************************************************************
```

Programda dikkat edilirse Timer1 harici clock girişi olan RC0 pin'i giriş olarak ayarlanmıştır. Ayrıca ana fonksiyon altında önceden genelde kullanılmadığı için kapatılan Timer1 birimi bu sefer kapatılmamıştır (Önceden kapatmak için setup_timer_1(T1_DISABLED); komutu kullanılırdı). Programda `enable_interrupts(INT_CCP1);` ve `enable_interrupts(INT_CCP2);` komutlarıyla CCP1 ve CCP2 kesmeleri aktif hale getirilmiştir. `enable_interrupts(GLOBAL);` komutu ile de tüm aktif edilen kesmelere izin verilmiştir. Ardından `setup_ccp1(CCP_COMPARE_INT);` ve `setup_ccp2(CCP_COMPARE_INT);` komutları ile CCP1 ve CCP2 birimlerinin COMPARE (karşılaştırma) modunda çalışması sağlanmıştır. COMPARE modu Timer1

zamanlayıcı/sayıcı birimini kullandığından bu birim ile ilgili ayarları belirtmek gerekir. Bu nedenle `setup_timer_1(T1_EXTERNAL_SYNC | T1_DIV_BY_1);` komutu ile Timer1 biriminin harici kaynaktan saat sinyali alacağını (Komuttaki T1_EXTERNAL_SYNC kısmı ile) ve Timer1 bölme oranının 1 olduğu (Komuttaki T1_DIV_BY_1 kısmı ile) belirtilmiştir. COMPARE modunda CCPRx kaydedicisinin içindeki değer ile Timer1 kaydedicisi olan TMR1'in içindeki değer sürekli olarak karşılaştırılmakta ve eşitlik olduğunda kesme meydana gelmekteydi. CCPRx kaydedicisi 16 bit'lik olduğundan CCPRxH:CCPRxL şeklinde 8 bit'lik iki kaydedici ile ifade edilir. Bu nedenle `ccp_1_HIGH=0x00;` komutu ile CCPR1H kaydedicisine heksadesimal 00h ve `ccp_1_LOW=0x05;` komutu ile de CCPR1L kaydedicisine heksadesimal 05h değeri yüklenmiştir. Sonuçta 16 bit'lik CCPR1 kaydedicisine desimal 5 değeri yüklenmiştir. Aynı mantıkla `ccp_2_HIGH=0x00;` komutuyla CCPR2H kaydedicisine heksadesimal 00h ve `ccp_2_LOW=0x0A;` komutu ile de CCPR2L kaydedicisine heksadesimal 0Ah (desimal olarak 10) değeri yüklenmiştir.

- CCS C'de; 16 bit'lik CCPR1 kaydedicisinin 8 bit'lik yüksek değerlikli kısmını tutan CCPR1H kaydedicisi CCP_1_HIGH, düşük değerlikli 8 bit'ini tutan CCPR1L kaydedicisi CCP_1_LOW olarak tanımlanmıştır. Aynı şekilde CCS C'de; CCPR2H kaydedicisi CCP_2_HIGH, CCPR2L kaydedicisi CCP_2_LOW olarak tanımlandığını hatırlayınız.

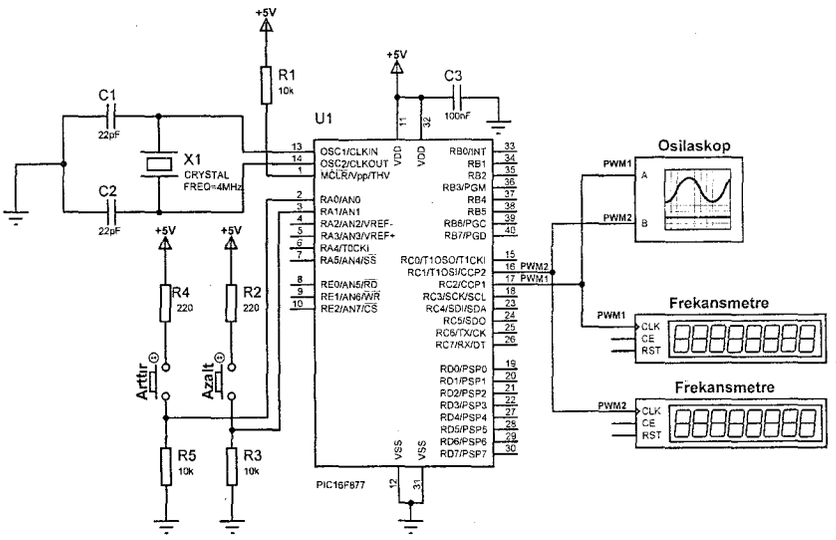
Timer1 kaydedicisi olan TMR1'in RC0/T1OSO/**T1CKI** pin'inden gelen saat sinyalinin her yükselen kenarında bir artacağı hatırlanmalıdır. Buna göre 5 kez butona basılıp bırakılınca yani TMR1 değeri 5 olunca TMR1 içeriği ile CCPR1 kaydedicisi birbirine eşit olur ve CCP1 kesmesi meydana gelir. CCP1 kesme fonksiyonunda `output_toggle(pin_b0);` komutu ile RB0 pin'ine bağlı LED'in durum değiştirmesi sağlanmıştır (yanıyorsa söner, sönük ise yanar). Yine butona basma bırakma sayısı 10 (heksadesimal olarak A)olunca TMR1 içeriği 10 olacaktır. Dolayısıyla TMR1 ve CCPR2 kaydedicisi içeriği birbirine eşit olmuş olur ve CCP2 kesmesi meydana gelir. CCP2 kesme fonksiyonuna da baktığımızda `set_timer1(0);` komutu ile ilk başta TMR1 içeriği sıfırlanmıştır. Ardından `output_toggle(pin_b1);` komutu ile RB1 pin'ine bağlı LED'in durum değiştirmesi sağlanmıştır. CCP2 kesme fonksiyonunda TMR1 içeriği sıfırlanarak devamlı olarak 5 kez butona basmada CCP1 kesmesinin, 10 kez butona basmada CCP2 kesmesinin meydana gelmesi sağlanmıştır. Eğer TMR1 içeriği CCP1 kesmesinde yapılsaydı, hiçbir zaman TMR1 değeri 10'a ulaşamayacaktı. Çünkü her 5 kez butona basmada TMR1 içeriği sıfırlanacaktı. Bu nedenle TMR1 içeriği CCP2 kesmesi fonksiyonunda sıfırlanmıştır. `while(1);` komutu ile de sonsuz döngü yaratılmış ve programın sonsuza kadar bu şekilde çalışması sağlanmıştır.

11.9. PWM MODU UYGULAMASI-1

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada CCP birimlerinin PWM modu özelliği incelenmiştir. Uygulamada iki PWM çıkışının (RC1/CCP2, RC2/CCP1) ikisi de kullanılmıştır. **PWM sinyallerinin frekansı sabit kalarak görev çevrimi sürelerinin değiştirilmesi işlemi incelenmiştir.** İki çıkışa da Şekil-11.8'de görüldüğü gibi osilaskop ve frekansmetre bağlanarak çıkış sinyal şekilleri görülebilir ve frekans değerleri ölçülebilir. Devrede RA0 ve RA1 girişlerine 2 buton bağlanmıştır. RA0 butonu PWM sinyallerinin görev çevrimi (duty cycle) süresine arttırmak, RA1 girişine bağlanan buton ise PWM sinyallerinin görev çevrimi süresini azaltmak için kullanılmıştır.

Devreyi PIC PROG/DEKA 16F87x deneme kartında denerseniz, RC2 pin'ine Buzzer bağlı olduğundan ve PWM görev çevrimi süresi değişirken PWM sinyalinin ortalama gerilim değeri değiştiğinden Artır ve Azalt butonlarına bastığınızda Buzzer'dan çıkan sesin değiştiğini inceleyebilirsiniz.



Şekil-11.8. *PWM modu uygulaması-1 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile PWM Modu Uygulaması-1
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
```

```
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

int i=10; // Tamsayı tipinde değişken tanımlanıyor

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı

   set_tris_a(0x03);      // RA0 ve RA1 pin'leri giriş
   set_tris_c(0x00);      // C port'u komple çıkış

   setup_ccp1(CCP_PWM);   // CCP1 birimi PWM çıkışı için ayarlandı
   setup_ccp2(CCP_PWM);   // CCP2 birimi PWM çıkışı için ayarlandı

   setup_timer_2(T2_DIV_BY_16,170,1); // Timer2 ayarları yapılıyor

   set_pwm1_duty(i); // PWM1 çıkışı görev çevrimi belirleniyor
   set_pwm2_duty(i); // PWM2 çıkışı görev çevrimi belirleniyor

   while(1) // Sonsuz döngü
   {
      if (input(pin_a0)) // Eğer RA0 girişine bağlı butona basılırsa
      {
         delay_ms(20);          // Buton arkı önleme gecikmesi
         while(input(pin_a0)); // Butondan el çekile kadar bekle
         i+=3;                  // i=i+3 anlamındadır.

         if (i>130) // i değeri 130'dan büyükse i değeri 130 olsun
            i=130;

         set_pwm1_duty(i); // PWM1 çıkışı görev çevrimi belirleniyor
         set_pwm2_duty(i); // PWM2 çıkışı görev çevrimi belirleniyor
      }

      if (input(pin_a1)) // Eğer RA1 girişine bağlı butona basılırsa
      {
         delay_ms(20);          // Buton arkı önleme gecikmesi
         while(input(pin_a1)); // Butondan el çekile kadar bekle
         i-=3;                  // i=i-3 anlamındadır.

         if (i<10) // i değeri 10'dan küçük ise i değeri 10 olsun
            i=10;
```

```
            set_pwm1_duty(i); // PWM1 çıkışı görev çevrimi belirleniyor
            set_pwm2_duty(i); // PWM2 çıkışı görev çevrimi belirleniyor
        }
    }
}
/*****************************************************************
```

Programda `setup_ccp1(CCP_PWM);` ve `setup_ccp2(CCP_PWM);` komutlarıyla PIC'in CCP birimlerinin PWM için kullanılacağı belirtiliyor. PWM birimi Timer2 zamanlayıcı/sayıcı birimini kullandığından `setup_timer_2(T2_DIV_BY_16,170,1);` komutu ile Timer2 ayarları yapılmıştır. Bu ayarlar PWM frekansının frekansını belirler. Oluşturulan PWM sinyalinin periyodunun ve frekansının hesaplanması aşağıda verilmiştir.

$$T_{PWM}=T_{OSC}.4.(PR2+1).(TMR2\ \text{bölme oranı})=\frac{1}{f_{OSC}}.4.(PR2+1).(TMR2\ \text{bölme oranı})$$

$$=\frac{1}{4000000Hz}x4x(170+1)x16=0.00000025snx4x171x16=0.002736sn$$

$$f_{PWM}=\frac{1}{T_{PWM}}=\frac{1}{0.002736sn}\quad 365.5Hz$$

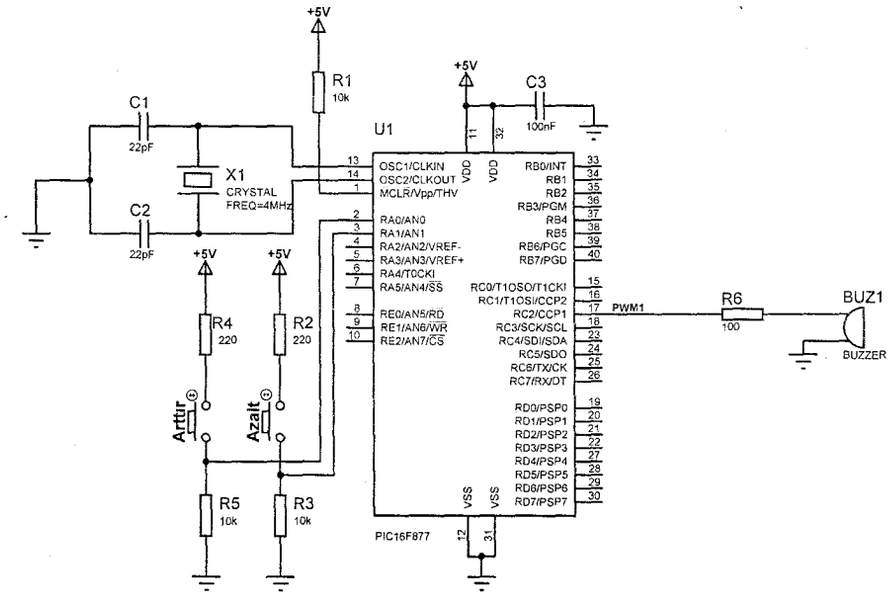
Ardından `set_pwm1_duty(i);` ve `set_pwm2_duty(i);` komutları ile PWM sinyallerinin görev çevrimi süreleri verilmiştir. Programın başında `int i=10;` komutu ile i değişkeni tanıtıldığından, ilk anda görev çevrimi değeri olarak 10 gönderilir. Sonsuz döngüde `if (input(pin_a0))` komutu ile RA0 pin'ine bağlı butona basılıp basılmadığı kontrol edilmiştir. Bu butona basıldığında ilk başta `delay_ms(20);` komutu ile buton arklarına önlemek için gerekli süre verilmiş ve ardından `while(input(pin_a0));` komutu ile de kullanıcının butondan elini çekene kadar başka bir işlem yapılmaması sağlanmıştır. Butondan el çelilince `while` döngüsünden çıkılacağından `i+=3;` komutu işlem görür. Bu komut `i=i+3;` ifadesi ile aynıdır. Bu komut ile görev saykıl süresini tutan i değişkeni değeri 3 arttırılmıştır. Daha önceden belirtildiği gibi PWM görev çevrimi süresi PWM periyodu süresinden daha fazla olmamalı. Bu nedenle `if (i>130)` komutu ile i değişkeni değeri kontrol edilmekte ve `i=130;` komutu ile de i değerinin en fazla 130 olması sağlanmıştır. Sonuçta hesaplanan i değişkeni değeri `set_pwm1_duty(i);` ve `set_pwm2_duty(i);` komutları ile PWM sinyalinin yeni görev çevrimi değerini belirlemiş olmaktadır.

Programda daha sonra RA1 pin'ine bağlı butona basılıp basılmadığı kontrol edilmiştir. Bu butona basıldığında da yukarıdaki işlemlerin aynısı yapılmış sadece bu butona her basışta i değişkeni değerinin `i-=3;` komutu ile 3 azaltılması sağlanmıştır. Ayrıca `if (i<10)` ve `i=10;` komutlarıyla i değerinin en az 10 olması sağlanmıştır.

11.10. PWM MODU UYGULAMASI-2

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada PWM sinyalinin frekansı butonlarla arttırılmak ve azaltılmak üzere değişimi sağlanmıştır. Uygulamada CCP1 birimi kullanılmıştır. RC2 pin'ine bir buzzer bağlanmıştır. Oluşturulan PWM sinyali buzzer'a gelmektedir. PWM sinyalinin frekansının değişimi ile buzzer'dan çıkan ses melodisi değişmektedir.



Şekil-11.9. *PWM modu uygulaması-2 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile PWM Modu Uygulaması-2
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

int i=65; // Tamsayı tipinde değişken tanımlanıyor
/********* ANA PROGRAM FONKSİYONU********/
```

```
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı

   set_tris_a(0x03);    // RA0 ve RA1 pin'leri giriş
   set_tris_c(0x00);    // C port'u komple çıkış

   setup_ccp1(CCP_PWM);  // CCP1 birimi PWM çıkışı için ayarlandı
   setup_timer_2(T2_DIV_BY_16,i,1); // Timer2 ayarları yapılıyor
   set_pwm1_duty(70);  // PWM1 çıkışı görev çevrimi belirleniyor

   while(1) // Sonsuz döngü
   {
      if (input(pin_a0)) // Eğer RA0 girişine bağlı butona basılırsa
      {
         delay_ms(20);          // Buton arkı önleme gecikmesi
         while(input(pin_a0)); // Butondan el çekile kadar bekle
         i+=5;                  // i=i+5 anlamındadır.

         if (i>250) // i değeri 250'dan büyükse i değeri 250 olsun
            i=250;
         setup_timer_2(T2_DIV_BY_16,i,1);//Timer2 ayarları yapılıyor
      }

      if (input(pin_a1)) // Eğer RA1 girişine bağlı butona basılırsa
      {
         delay_ms(20);          // Buton arkı önleme gecikmesi
         while(input(pin_a1)); // Butondan el çekile kadar bekle
         i-=5;                  // i=i-5 anlamındadır.

         if (i<65) // i değeri 65'den küçük ise i değeri 65 olsun
            i=65;
         setup_timer_2(T2_DIV_BY_16,i,1);//Timer2 ayarları yapılıyor
      }
   }
}
//*****************************************************************
```

Bu programda da PWM uygulaması-1 programında anlatılan işlemlerin neredeyse aynısı kullanılmıştır. Fark olarak ilk programda PWM frekansı sabitti ve butonlarla sadece görev çevrimi süresi değiştiriliyordu. Bu programda ise görev çevrimi süresi `set_pwm1_duty(70);` komutu ile belirlenmiş ve sabit bırakılmıştır. Fakat programda ilgili yerlerde `setup_timer_2(T2_DIV_BY_16,i,1);` komutuyla PWM frekansı "i" değişkeni değerine göre değiştirilmiştir. RA0 pin'ine bağlı butona her basılışta `i+=5;` komutu ile i değişkeni değeri 5 arttırılmıştır. Aynı mantıkla RA1 pinine bağlı butona her basılışta `i-=5;` komutu ile i değişkeni değeri 5 azaltılmıştır. `if (i>250)` ve `i=250` komutları ile i değişkeni değerinin en fazla

250 olması sağlanmıştır. `if (i<65)` ve `i=65` komutları ile de i değişkeni değerinin en az 65 olması sağlanmıştır. Butona basıldığı algılandığında her iki buton döngüsünde de `setup_timer_2(T2_DIV_BY_16,i,1);` komutu ile PWM frekansı yeniden belirlenmiştir.

Devrenizi Proteus-Isis programında denerken Buzzer üzerine farenin sağ tuşu ile basarak seçtikten sonra farenin sol tuşu ile Buzzer üzerine tıklayarak ayar menüsünü açın. Menüde "Operating Voltage" kısmını 2V, "Load Resistance" kısmını ise 100 yapmanız benzetim (simülasyon) esnasında ortaya çıkabilecek çalışmama sorununu ortadan kaldıracaktır.

12. Bölüm

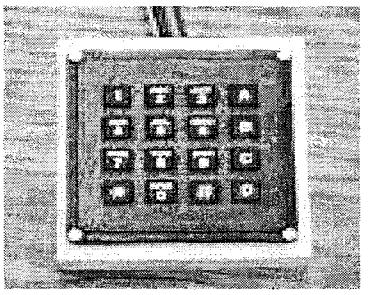
TUŞ TAKIMI (KEYPAD) UYGULAMALARI

- Tuş Takımı (Keypad) Hakkında Bilgi
- Tuş Takımı Uygulaması-1
- 74C922 Tuş Takımı Enkoder Entegresi
- Tuş Takımı Uygulaması-2 (74C922 İle)

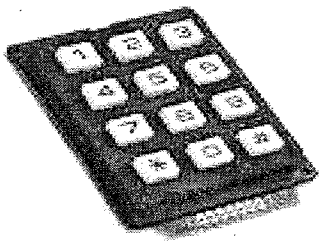
Bu bölümde tuş takımı diğer ifade ile keypad-klavye hakkında bilgiler verilmiştir. Bu bilgiler uygulamalar ile pratiğe dökülmüştür. Bölümde ilk başta program komutları ile bir tuş takımı tarama işlemleri ve uygulamaları incelenmiştir. Ayrıca 74C922 tuş takımı enkoder entegresi tanıtılmış ve bu entegre ile bir uygulama gerçekleştirilmiştir.

12.1. TUŞ TAKIMI (KEYPAD) HAKKINDA BİLGİ

Kontrol sistemlerinde dış dünyadan insanlar tarafından veri girişleri genellikle tuş takımı (keypad-klavye) ile yapılır. Tuş takımı butonlarla gerçekleştirilebileceği gibi çeşitli hazır tuş takımları piyasada bulunmaktadır. Tuş takımı isimlendirmelerinde ilk sayı sütun, ikinci sayı ise satır sayısını belirtir. Örneğin 4x3'lük bir keypad, 4 sütun ve 3 satırdır.

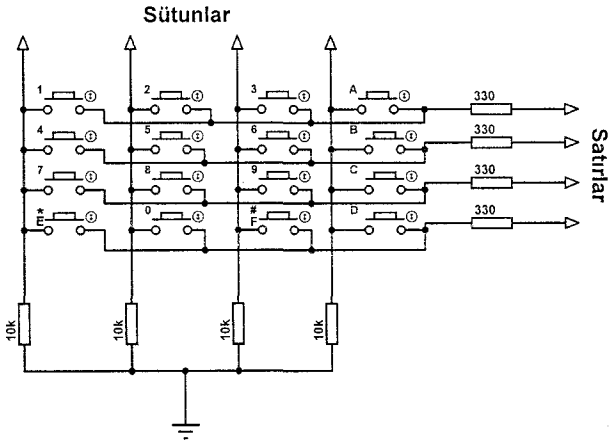


Şekil-12.1. *4x4 Tuş takımı.*



Şekil-12.2. *4x3 Tuş takımı.*

Tuş takımında hangi tuşa basıldığını bulmak için çeşitli yöntemler kullanılabilir. Bu yöntemlerden biri tarama yöntemidir. Şekil-12.3.'de butonlarla yapılmış 4x4 tuş takımı görülmektedir. Butonların bir ucu satır kısmına, bir ucu da sütun kısmına bağlıdır. Denetleyici ile tarama yapılırken **satırlar çıkış, sütunlar ise giriş olarak tanımlanır.**



Şekil-12.3. *Butonlarla yapılmış 4x4 tuş takımı.*

Sütunlarda hep lojik-0 (GND-şase) vardır. Hangi tuşa basıldığını anlamak için önce satırlardan biri lojik-1 diğerleri lojik-0 yapılır. Sonra sütunlar okunur, hangi giriş lojik-1 ise o satıra ait sütundaki tuşa basılmış demektir. İstenen tuşa hangi değerin verileceği programcıya aittir. Şekil-12.3'teki bağlantıda 1. satır lojik-1 diğer satırlar lojik-0 iken, 1.satırda 3 nolu tuşa basıldığında 3.sütunda lojik-1 bilgisi okunur. Böylece basılan tuş bulunabilir. Tabi ki 1.satırda değil de diğer satırlardaki tuşlara basılmışsa algılama yapılamaz. Bu nedenle 1.satırdan sonra sıra ile bir satır lojik-1, diğer satırlar lojik-0 yapılarak tüm sütunlar okunur. Bahsedilen tarama işlemi bu şekilde yapılmaktadır. Tablo-12.1'de, Şekil-12.3'teki tuş takımına göre uyarlanmış tarama bilgileri verilmiştir.

Butona basıldığında ve bırakıldığında bir ark (parazit) meydana gelir. Buna tuş sıçraması da (key debounce) denilir. Şekil-12.4'te örnek bir tuş sıçraması görülmektedir. Bu sıçramayı önlemek için programda gerekli önlemler alınmalıdır. Tedbir olarak butona basıldıktan sonra 15-20 msn gecikme verilmesi gerekir veya butondan el çekilene kadar içinden çıkılmayacak bir döngü kurulmalıdır. Ayrıca tuş takımında aynı anda iki tuşa birden basılabilir. Bu gibi durumlarda hangi tuşun geçerli olacağı programla belirtilerek istenmeyen durumlar önlenmelidir.

Şekil-12.4. *Tuş basılıp bırakılmasında oluşan parazit.*

Satırlar					Sütunlar					Basılan Tuş
4	3	2	1		4	3	2	1		
0	0	0	1		0	0	0	1		1
0	0	0	1		0	0	1	0		2
0	0	0	1		0	1	0	0		3
0	0	0	1		1	0	0	0		A
0	0	1	0		0	0	0	1		4
0	0	1	0		0	0	1	0		5
0	0	1	0		0	1	0	0		6
0	0	1	0		1	0	0	0		B
0	1	0	0		0	0	0	1		7
0	1	0	0		0	0	1	0		8
0	1	0	0		0	1	0	0		9
0	1	0	0		1	0	0	0		C
1	0	0	0		0	0	0	1		*, E
1	0	0	0		0	0	1	0		0
1	0	0	0		0	1	0	0		#, F
1	0	0	0		1	0	0	0		D

Tablo-12.1. *Tuş takımı tarama bilgileri.*

12.2. TUŞ TAKIMI UYGULAMASI-1

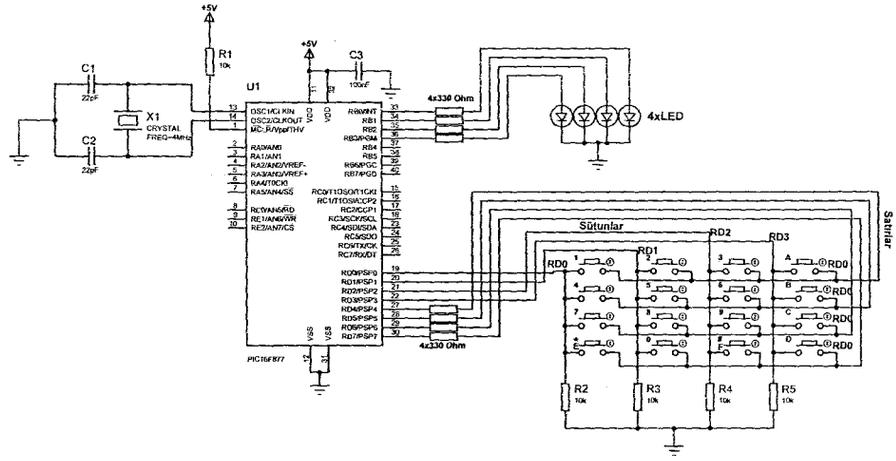
Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada butonlarla yapılmış 4x4 bir tuş takımı uygulaması yapılmıştır. Tuş takımı D port'una bağlanmıştır. B port'unda RB0, RB1, RB2 ve RB3 pin'lerine LED bağlanmıştır. Tuş takımından basılan tuşların ikili karşılıkları LED'lerde gösterilmektedir. Tuş takımı için programda bir fonksiyon yazılmıştır. Programda tuş takımında bir tuşa basıldıktan sonra tuşun değeri B port'una bağlı LED'lerde görünecektir.

Programda her satır sırayla lojik-1 yapılmış ve ardından sütun girişleri kontrol edilmiştir. Hangi sütun lojik-1 ise o sütundaki butona basılmış demektir. Örneğin 2. satır lojik-1 yapıldığında, 2. sütun girişi lojik-1 ise demek ki 5 tuşuna basılmış demektir.

Programda yapılan fonksiyonda tarama sonucu basılan tuş değeri "tus" değişkenine aktarılmıştır. "tus" değişkeni kullanılarak elde edilen sonuç istenen şekilde kullanılır. Tuş takımında birden fazla tuşa basılması için gerekli önlem alınmamıştır. İstenirse bu gibi durumlarda basılan en son tuşun geçerli olması veya ilk basılan tuştan el

kalkana kadar, diğer basılan tuşların geçerli olmaması sağlanabilir. İstenen amaca göre program değiştirilebilir.



Şekil-12.5. *Tuş takımı uygulaması-1 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Tuş Takımı Uygulaması-1
      PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(d) //Port yönlendirme komutları C port'u için geçerli

#byte   portb=0x06   // B port'u "portb" kelimesine eşitleniyor.

#define sut1   pin_d0 // sut1 ifadesi pin_d0 ifadesine eşitleniyor
#define sut2   pin_d1 // sut2 ifadesi pin_d1 ifadesine eşitleniyor
#define sut3   pin_d2 // sut3 ifadesi pin_d2 ifadesine eşitleniyor
#define sut4   pin_d3 // sut3 ifadesi pin_d2 ifadesine eşitleniyor

#define sat1   pin_d4 // sat1 ifadesi pin_d4 ifadesine eşitleniyor
#define sat2   pin_d5 // sat2 ifadesi pin_d5 ifadesine eşitleniyor
#define sat3   pin_d6 // sat3 ifadesi pin_d6 ifadesine eşitleniyor
#define sat4   pin_d7 // sat4 ifadesi pin_d7 ifadesine eşitleniyor

char tus=0; // karakter tipinde değişken tanımlanıyor
```

```
//*********** Keypad Tarama Fonksiyonu *********
char keypad_oku() // Fonksiyon ismi
{
   output_d(0x00); // D port'u çıkışı sıfırlanıyor

   output_high(sat1); // 1. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=1; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=2; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=3; }
   if (input(sut4))   // 4. sütun okunuyor
      { delay_ms(20); tus=0xA; }
   output_low(sat1); // 1. satır lojik-0 yapılıyor

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

   output_high(sat3); // 3. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=7; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=8; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=9; }
   if (input(sut4))   // 4. sütun okunuyor
      {delay_ms(20); tus=0x0C; }
   output_low(sat3); // 3. satır lojik-0 yapılıyor

   output_high(sat4); // 3. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=0xE; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=0; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=0xF; }
   if (input(sut4))   // 4. sütun okunuyor
      {delay_ms(20); tus=0xD; }
   output_low(sat4); // 3. satır lojik-0 yapılıyor

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

```
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00);  // B port'u komple çıkış
   set_tris_d(0x0F);  // D portunun Yüksek değerlikli 4 bit çıkış,
                      // düşük değerlikli 4 bit giriş

   output_b(0x00); // İlk anda B port'u çıkışı sıfırlanıyor

   while(1)  // Sonsuz döngü
   {
      portb=keypad_oku(); // Basılan tuş değerini B port'una aktar
   }
}
//*****************************************************************
```

Programda `#byte portb=0x06` komutu ile B port'unun adresi `(0x06)` `"portb"` ifadesine eşitlenmiştir. Yani bu komuttan sonra artık `"portb"` ifadesi direkt olarak B port'u değerini temsil etmektedir. Programda `#define sut1 pin_d0` komutu ile `"sut1"` ifadesi pin_d0'a yani RD0 pin'ine eşitlenmiştir. Diğer sütun ve satır ifadeleri de D port'unun ilgili pin'lerine aynı şekilde eşitlenmiştir.

`char tus=0;` komutu ile karakter tipinde `"tus"` isminde değişken tanımlanmış ve ilk değer olarak sıfır (0) verilmiştir. Tuş takımı tarama fonksiyonu olarak `char keypad_oku()` komutu ile `keypad_oku()` isminde fonksiyon tanımlanmıştır. Fonksiyonun başındaki `"char"` ifadesi, **fonksiyonun geri dönüş değeri olduğunu** ve bu değerin karakter tipinde olduğunu belirtir. Fonksiyonda ilk başta `output_d(0x00);` komutu ile D port'una bağlı sütunların hepsi lojik-0 yapılmıştır. Daha sonra `output_high(sat1);` komutuyla sırayla her satır lojik-1 yapılmış ve `if (input(sut1))` komutu ile de sütun girişleri okunmuştur. Eğer sütun girişi "1" olursa if fonksiyonu döngüsü işlem görecektir. if fonksiyonu içinde, `delay_ms(20);` komutu ile tuş hatalarını önlemek için bir gecikme konulmuş ve ardından `tus=1;` komutu ile de basılan tuş değeri "tus" değişkenine aktarılmıştır. Burada `tus=` ifadesine istenen değer yazılarak hangi tuşun neye karşılık geleceği istenen şekilde değiştirilir. İlk satırda 4 sütunun taraması aynı mantıkla yapıldıktan sonra `output_low(sat1);` komutu ile 1. satır lojik-0 yapılmıştır. Daha sonra aynı işlemler diğer satırlar için yapılmıştır. Fonksiyonda en son olarak da `return tus;` komutu ile fonksiyonun "tus" değişkeni değeri ile geri dönmesi sağlanmıştır.

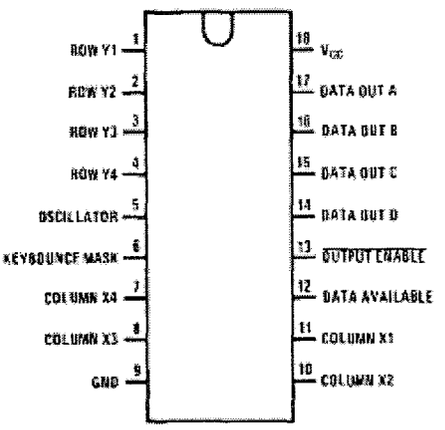
Dikkat edilmesi gereken bir nokta da, fonksiyon bir geri dönüş değeri içerdiğinden fonksiyon tanımlarken `char sutun_tara ()` komutunda başta "`char`" kısmı ile fonksiyonun geri dönüş değeri olduğunu ve türünün karakter olduğunu belirtmemizdir. Fonksiyonun geri dönüş değeri olmasaydı başına "`void`" koymamız gerekirdi.

Ana fonksiyondaki sonsuz döngüde `portb=keypad_oku();` komutu ile tuş takımı tarama fonksiyonu çağrılmış ve bu fonksiyonda tuş değerinin tutulduğu "tus" değişkeni değeri B port'una gönderilmiştir.

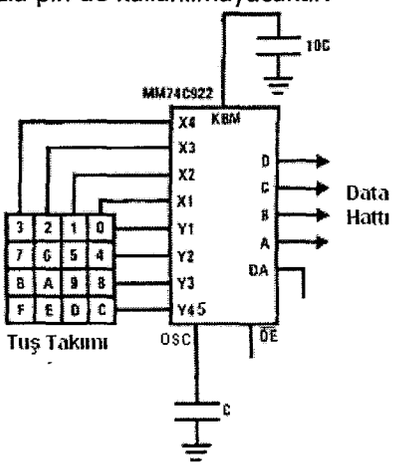
Tuş takımı tarama fonksiyonunu kendi amaçlarınıza ve ihtiyaçlarınıza göre değiştirebilirsiniz.

12.3. 74C922 KEY ENKODER ENTEGRESİ

Tuş tarama işlemi bir önceki uygulamada yapıldığı gibi kullanıldığında denetleyiciyi her zaman meşgul eder. Yani tuş tarama işleminin sürekli olarak yapılması gerekmektedir. Aynı anda başka işlemlerin de yapılacağı bir uygulamada bu yöntem sorunlara yol açabilir. Aynı zamanda önceki uygulamada tuş tarama işlemi için 8 adet pin kullanılmaktaydı, bu da port ihtiyacının fazla olduğu uygulamalarda sıkıntıya yol açabilir. Bu nedenle tuş takımı tarama işlemlerinde 74C922 16 Key Encoder entegresi kullanılabilir. Bu entegre sayesinde tarama işlemi için denetleyici tümüyle meşgul edilmeyecek ve aynı zamanda fazla pin de kullanılmayacaktır.



Şekil-12.7. *74C922 Key encoder entegresi pin diyagramı.*



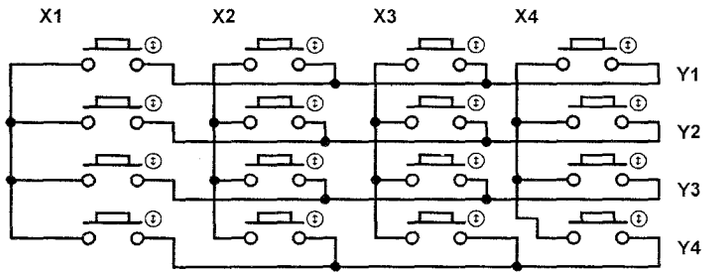
Şekil-12.8. *74C922 Entegresi genel bağlantısı.*

74C922 entegresi 3V ile 15V çalışma gerilim aralığında çalışabilir. Genel besleme gerilimleri 5V, 10V ve 15V'tur. Klavye satır hatları Y girişlerine (ROW), sütun hatları

ise X girişlerine (COLUMN) bağlanır. $\overline{OE}$ entegre seçme ucudur ve lojik-0'da aktiftir. Bu uca 0 verildiğinde entegre çıkışı aktif yapılmış demektir. "Data Available (DA)" ucu ise, klavyede herhangi bir tuşa basıldığında çıkışı lojik-1 olur. Böylece klavyeden bir tuşa basıldığını bize entegre haber verir. $\overline{OE}$ ucu aktif olmasa bile DA ucu herhangi bir tuşa basıldığında aktif olur, fakat $\overline{OE}$ ucu aktif olana kadar basılan tuşun değeri data çıkış uçlarında (A, B, C, D) görülmez. "Keybounce Mask-KBM" ucu ise, tuşa basıldıktan sonra oluşan arkların önlenmesi için ne kadar bir süre beklenmesi gerekeceğini belirleyen bir kontrol ucudur. Bu uca bağlanacak kondansatörün değeri bekleme süresini belirler. Tarama osilatör ucuna (Oscillator, pin 5) bir osilatör bağlanabileceği gibi sadece kondansatörde bağlanabilir. Kondansatör bağlanması durumunda Şekil-12.8'de görüldüğü gibi KBM ucuna bağlanan kondansatör değeri, OSC ucuna bağlanan kondansatör değerinin 10 katı olmalıdır. 74C922 entegresi toplam 16 tuş için 4 bit'lik değerler üretir. Şekil-12.8. ve Tablo-12.2. birlikte incelendiğinde hangi satır ve sütundaki (X sütun, Y satır) tuşa basıldığında entegre tarafından üretilen değer (kesişme olan tuş) görülebilir. Program vasıtasıyla üretilen bu değerlere istenen karakter karşılığı verilebilir.

Tuş	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pozisyonu	Y1,X1	Y1,X2	Y1,X3	Y1,X4	Y2,X1	Y2,X2	Y2,X3	Y2,X4	Y3,X1	Y3,X2	Y3,X3	Y3,X4	Y4,X1	Y4,X2	Y4,X3	Y4,X4
DATA ÇIKIŞI	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Tablo-12.2. *74C922 entegresi doğruluk tablosu.*

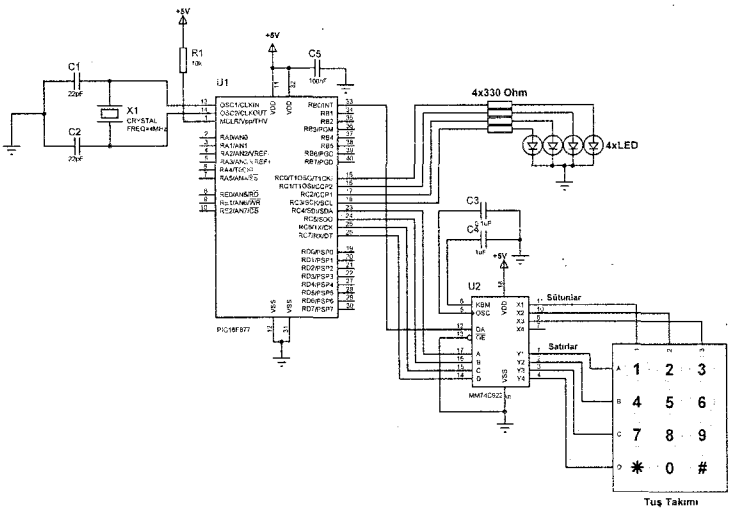


Şekil-12.9. *Satır ve Sütun bağlantıları.*

12.4. TUŞ TAKIMI UYGULAMASI-2 (74C922 İLE)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada tuş takımını taramak için 74C922 tuş çözücü entegresi kullanılmıştır. Böylece hem kullanılan pin sayısı azalmış hem de tuş tarama için sürekli olarak programın meşgul edilmesi önlenmiştir. Devre şemasında dikkat edilirse 74C922 entegresinin DA ucu PIC'in RB0/INT pin'ine bağlanmıştır. Denetleyici RB0/INT harici kesmesine ayarlanmıştır. 74C922 entegresinin DA ucu (Data Available) herhangi bir tuşa basılınca "1" olur. Bu şekilde bir tuşa basılana kadar denetleyicinin tuş takımı ile bir iş yapmasına gerek kalmaz. Ne zaman ki tuşa basılır, DA ucu "1" olur, dolayısıyla dış kesme meydana gelir. İşte o zaman denetleyici kesme fonksiyonuna giderek tuş takımı ile ilgili işlemleri yapar. Aksi durumda denetleyici diğer komutlarını işlemeye devam eder. Bu sayede denetleyicinin boşa tuş tarama işlemi yapması önlenmiş olur. Dikkat edilirse tuş önceki uygulamada 4x4 tuş takımı için 8 uç gerekliydi. Bu uygulamada ise 74C922 çıkışlarından alınan 4 uç tuş tarama bilgisini almak için yeterlidir. Devrede kullanılan 3x4 tuş takımı yerine 4x4 tuş takımı kullanılabilir. Devredeki tuş takımında bulunan "*" ve "#" tuşlarına "0x0F" değeri atanmıştır.



Şekil-12.10. *Tuş takımı uygulaması-2 devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Tuş Takımı Uygulaması-2 (74C922 İle)
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
```

```
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli

#byte   portc=0x07   // C port'u "portc" ismine eşitleniyor.

char const keys[] ={1,2,3,0,    // Gelen bilgilere göre gösterilmesi
                    4,5,6,0,    // istenen karakterler
                    7,8,9,0,
                    0xF,0,0xF,0 };

//*********** Dış Kesme Fonksiyonu Tanımlanıyor ********
#int_ext            // Dış kesme fonksiyonu
void dis_kesme ()
{
   portc=keys[portc>>4];   // C port'una C port'unun okunan yüksek
                           // değerlikli 4 bit'i aktarılıyor
}


//********** ANA PROGRAM FONKSİYONU*******
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x01);   // B port'u komple çıkış
   set_tris_c(0xF0);   // C portunun Yüksek değerlikli 4 bit çıkış,
                       // düşük değerlikli 4 bit giriş

   ext_int_edge(L_TO_H);  // INT_EXT dış kesmesinin yükselen kenarda
                          // aktif olacağını belirtir

   enable_interrupts(INT_EXT);  // Dış kesme aktif
   enable_interrupts(GLOBAL);   // Aktif edilen kesmelere izin ver

   portc=0;  // Başlangıçta C port'u çıkışı sıfırlanıyor

   while(1);  // Sonsuz döngü
}
//*****************************************************************
```

Programda `#byte portc=0x07` komutu ile "portc" ifadesi C port'una eşitlenmiştir (0x07 adresi C port'unun bellekteki adresidir). Ardından `char const keys[] =...` komutu ile tuş takımındaki tuşların değerlerini içeren dizi tanımlanmıştır. Komuttaki

const kelimesi bu dizi elemanlarının değişken değil sabit olduğunu belirtir. Dizideki değerler değiştirilerek istenen tuşa basılınca görünmesi istenen değer değiştirilebilir. Dizide dikkat edilirse 4. sütun tuş değerleri hep "0" yapılmıştır. Bunun nedeni kullandığımız tuş takımının 3 sütun 4 satır olmasındandır. **Eğer 4 sütun 4 satır bir tuş takımı kullansaydık bu "0" değerleri yerine 4. sütunda bulunan tuş değerlerini yazacaktık.**

`#int_ext` ve `void dis_kesme ()` komutları ile RB0/INT dış kesme fonksiyonu tanımlanmıştır. Fonksiyonda `portc=keys[portc>>4];` komutu ile C port'una, C port'unun yüksek değerlikli 4 bit'inden (RC4, RC5, RC6 ve RC7) okunan değer aktarılmıştır. Okunan değer C port'unun yüksek değerlikli 4 bit'inden gelmektedir. Fakat LED'ler C port'unun düşük değerlikli 4 bit'ine (RC0, RC1, RC2 ve RC3) bağlanmıştır. Bu nedenle okunan değer `portc>>4` komutu ile 4 kez sağa kaydırılarak RC4, RC5, RC6, RC7 pin'lerinden okunan değerlerin RC0, RC1, RC2, RC3 pin'lerine karşılık gelmesi sağlanmıştır. Örneğin tuş takımından "3" tuşuna basıldığında 74C922 entegresinden gelen DCBA uçları değeri 0010 olur (Bu değeri Tablo-12.2'de görebilirsiniz. Tablo-12.2'deki tuş değerleri Şekil-12.8'de bağlantısı verilen tuş takımı içindir. Siz bu tuş takımındaki tuşlara istediğiniz değeri program aracılığı ile atayabilirsiniz). C port'unun değeri o an 0010xxxx olur. `portc>>4` komutu ile 4 kez sağa kaydırılınca bu değer 00000010 değerini alır. Bu değerin desimal karşılığı 2'dir. Dolayısıyla `portc=keys[portc>>4];` komutu `portc=keys[2];` halini alır. keys dizisinin 2. elemanı "3"'tür (Dizilerde, dizi eleman numaralandırmasına 0'dan başlandığını hatırlayınız). Komut `portc=3;` halini alır ve C port'una bağlı LED'lerde "3" değerinin ikili sayı karşılığı görülür.

Programda `ext_int_edge(L_TO_H);` komutu ile dış kesmenin yükselen kenarda tetikleneceği belirtilmiştir. `enable_interrupts(INT_EXT);` komutu ile de dış kesme aktif hale getirilmiş ve `enable_interrupts(GLOBAL);` komutuyla da aktif edilen kesmelere izin verilmiştir. `while(1);` komutu ile oluşturulan sonsuz döngü ile program sürekli çalışmaya devam eder. İstenirse bu döngü içerisine başka komutlar yazılabilir. Çünkü denetleyici, bir tuşa basılana kadar kesme fonksiyonuna gitmeyecek ve boşta bekleyecektir.

Piyasada çeşitli tipte bulunan tuş takımlarının uçları farklı sıralarda olabilir. Bu nedenle tuş takımınızı devrede kullanmadan ve programınızı yazmadan önce tuş takımınızın uçlarını (hangileri hangi sütun ve satır diye) tespit etmeniz gerekmektedir.

13. Bölüm

KARAKTER LCD ve GRAFİK LCD UYGULAMALARI

- Paralel LCD
- Seri LCD Display
- CCS C Karakter Tabanlı Paralel LCD Kütüphanesi
- Paralel LCD Uygulaması
- Tuş Takımı ve LCD Uygulaması
- Kendi LCD Dosyamızı Oluşturma
- Kendi LCD Dosyamızla LCD Uygulaması
- LCD'de Türkçe ve Özel Karakter Oluşturma
- Özel Karakterli LCD Dosyası Oluşturma
- Özel Karakterli LCD Uygulaması
- Grafiksel LCD
- CCS C Grafik LCD Kütüphane Dosyaları
- Grafik LCD Uygulaması

Bu bölümde genel olarak LCD'ler hakkında bilgi verilmiştir. Karakter tabanlı paralel ve seri LCD'ler ile grafiksel tabanlı LCD incelenmiştir. CCS C kütüphanelerinden karakter tabanlı LCD ve grafiksel tabanlı LCD'ler için bulunan kütüphaneler tanıtılmış ve bu kütüphaneler ile çeşitli uygulamalar yapılmıştır. Karakter tabanlı LCD'ler için hazır kütüphane kullanımının yanında kendi LCD fonksiyonlarımızı nasıl oluşturacağımız hakkında ayrıntılı bilgi verilmiştir. Ayrıca LCD'de Türkçe karakter ile istenen özel karakterlerin nasıl oluşturulacağı da ayrıntılı olarak incelenmiştir.

13.1. PARALEL LCD

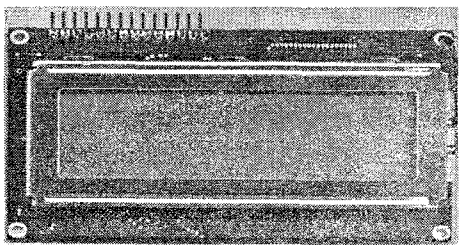
LCD kısaltması **L**iquid **C**rystal **D**isplay açılımının kısaltılmış halini belirtir. Türkçe olarak da Sıvı Kristal Görüntü Birimi olarak tanımlanmaktadır. Karakter tabanlı LCD'ler (metin LCD'ler) bize birçok harf, sembol ve karakterleri görüntüleme imkânı verirler. Grafik LCD'ler de istediğimiz çizimleri görüntülememizi sağlarlar. Karakter

tabanlı LCD'ler **5x7'lik (aslında 5x8'dir, fakat en alt kısım imleç içindir)** veya **5x10'luk** dot matris (nokta matris) hücrelerinden oluşurlar. Şekil-13.1'de 5x7'lik dot matris LCD'de gösterilen bir "R" karakteri görülmektedir. LCD'de karakterlerin gösterilmesi için 7 segment display uygulamasında incelendiği gibi tarama yöntemi kullanılır. Fakat LCD'de gösterilen karakterlerin tarama işlemi mikrodenetleyiciyi neredeyse tümüyle meşgul edebilir. Bu nedenle LCD içinde bu tarama işlemini yapan entegreler mevcuttur. LCD'ler isimlendirilirken satır ve sütun sayısına göre isimlendirilir. İsimlendirmede ilk sayı satır, ikinci sayı sütun sayısını belirtir. Örneğin 2x16 LCD, 2 satır ve 16 sütundan oluşmaktadır.

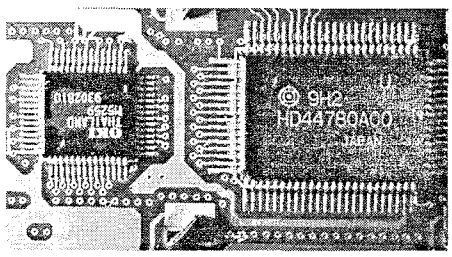
1	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	1	1	1	0
1	0	1	0	0
1	0	0	1	0
1	0	0	0	1
0	0	0	0	0

Şekil-13.1. *5x8 Dot Matris LCD'de "R" karakterinin görünümü.*

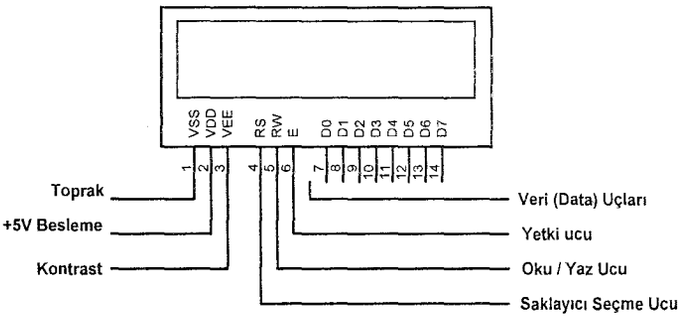
Piyasada bulunan LCD'ler genelde Hitachi firmasının **HD44780** entegresi ve buna uyumlu entegre içerir. Genelde bir karakter LCD'nin 14 adet ucu vardır. Normal beslemesine ek olarak LCD arka ışık besleme uçları da vardır. Bu nedenle bazı LCD'ler 16 uçludur. 16 uçlu LCD'lerde 2 uç LCD arka ışık beslemesi için gerekli + ve - besleme uçlarıdır. Bu uçlar genelde LCD'nin yan kısımlarında olurlar. Şekil-13.2'de bir karakter LCD görünümü ve Şekil-13.3'de de LCD'nin arka görünümü verilmiştir. Şekil-13.4'de ise LCD pin uçları görülmektedir. LCD, 4 veya 8 bit'lik denetleyici ile sürülebilir. 8 bit'lik veya 4 bit'lik veri bilgisi ile sürülebildiği gibi tek bit'lik veri bilgisiyle de sürülebilir.



Şekil-13.2. *2x16 Karakter LCD.*



Şekil-13.3. *LCD arka paneli.*



Şekil-13.4. *Karakter LCD pin isimleri.*

Şekil-13.3'te verilen karakter LCD pin isimleri ve numaraları genel olarak piyasada bulunan çoğu karakter LCD'ye uymaktadır. **Fakat tüm LCD'ler ile uyumlu değildir.** LCD'ler üzerinde pin numaraları sıralı şekilde 1, 2, 3, ... 14 veya 16'ya kadar belirtilmiş olmasına rağmen pin'ler farklı uçlara karşılık gelebilir. Örneğin her LCD'de 3 numaralı pin V_{EE}, 5 numaralı pin RW ucu değildir. LCD'nin arkasında bulunan LCD'nin kullandığı entegrenin kodu veya LCD arkasında yazan LCD modeli ile o LCD'nin teknik bilgi notlarına (datasheet) bakılmalı ve LCD uçları karşılıkları öğrenilmelidir.

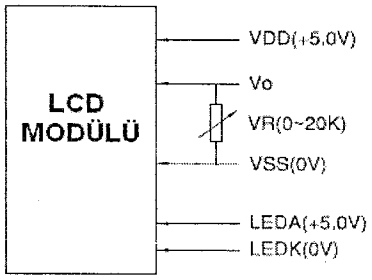
RS (Register Select) ucu, veri yolundan gelen bilginin komut veya veri bilgisi olup olmadığını LCD'ye bildirir. Eğer RS=0 ise D0-D7 uçlarından (veri yolu) gelen bilgi LCD tarafından komut olarak algılanır. RS=1 olursa veri hattındaki bilgi karakter bilgisi olarak algılanır.

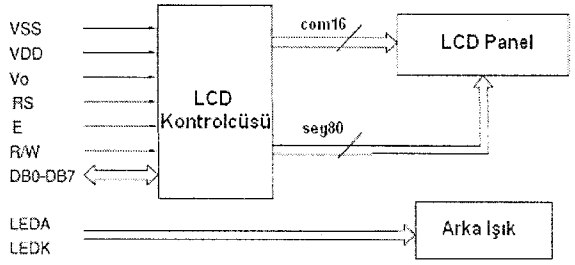
R/W (Read/Write) ucu, LCD'ye bilgi göndermek ve LCD'den bilgi almak için kullanılır. R/W=1 ise LCD'den bilgi okuma, R/W=0 ise LCD'ye bilgi gönderme işlemi yapılır.

E (Enable) ucu, yetki verme ucudur. Her işlemin (okuma, yazma vs.) gerçekleşmesi için bu ucun ilk başta lojik-1 daha sonra ise lojik-0 yapılması gereklidir **(düşen kenarda tetiklenir)**.

D0-D14 uçları, bu uçlar veri uçlarıdır. Bu uçlardan hem görünmesi istenen karakter verileri gönderilir, hem LCD'ye komut gönderilir hem de LCD'den veri alımı için kullanılır.

VDD, VSS, VEE(V0) uçları, bu uçlar LCD besleme uçlarıdır. V_{DD} ucu pozitif besleme ucudur. Bu uca +5V verilmelidir. V_{SS} ucu ise şase ucudur. V_{EE} veya V_0 uçları aynı anlamdadır. Bazı kaynaklarda V_{EE} bazı kaynaklarda V_0 olarak gösterilmektedir. Bu uç LCD'de görünen karakterin parlaklık (kontrast) ayar ucudur. Şekil-13.5'de V_{DD}, V_{SS} ve V_0 uçlarının bağlantısı görülmektedir. Şekilden görüldüğü gibi V_0 ve V_{SS} arasına genelde bir potansiyometre bağlanır (genelde 5K'lık potansiyometre yeterlidir). Potansiyometre yerine sabit bir direnç de bağlanabilir. **LEDA ve LEDK** uçları LCD arka ışıklandırması besleme uçlarıdır.



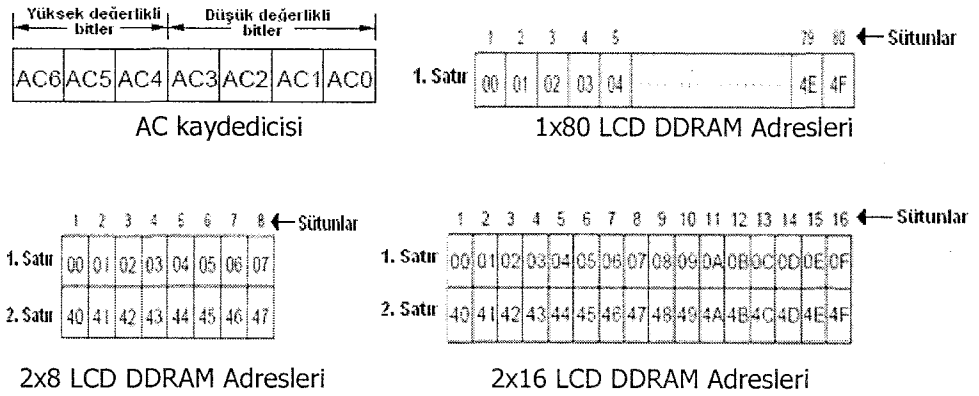


Şekil-13.5. *LCD Besleme uçları.* ***Şekil-13.6.*** *LCD Blok diyagramı.*

13.1.1. LCD DDRAM Belleği ve LCD Karakter Tablosu

Karakter tabanlı LCD blok diyagramında (Şekil-13.5) görüldüğü her LCD'nin bir kontrolcüsü vardır. Bu kontrolcü LCD işlemlerini yapmak için belli komutlara ve belleğe sahiptir. Bellek olarak LCD'ye gönderilen display'de gösterilmesi istenen karakter kodlarının tutulduğu geçici **veri belleği (Display Data RAM -DDRAM-)** vardır. DDRAM LCD'nin kaç satır ve sütuna sahip olduğuna bakmaksızın 80 adet (80x8 bit) karakter kapasitesine sahiptir. Display'de gösterilmek istenen karakter verileri DDRAM'a kaydedilir. Gönderilen karakter verileri DDRAM'a sırayla yerleştirilir. Veri adresi **AC adres sayıcı** (**A**dres **C**ounter- **AC**) ile otomatik olarak artar. Şekil-13.7'de AC adres sayıcı kaydedicisi bit'leri ve çeşitli karakter tabanlı LCD'lerin DDRAM bellek adresleri görülmektedir.

Örneğin 2x8 LCD'de 80 adet karakter adresini yerleştirmenin nasıl mümkün olduğunu düşünebilirsiniz. Bunun için DDRAM adresi sağa veya sola kaydırılır. Bahsedilen kaydırma işlemini Şekil-13.8'de görebilirsiniz. Şekilden görüldüğü gibi 2x8 LCD'de toplam 16 adetlik DDRAM yeri vardır. Diğer kalan adresler için (toplam 80 adet olduğunu hatırlayınız) DDRAM ya sola ya da sağa kaydırılır.



Şekil-13.7. *AC kaydedicisi ve çeşitli LCD DDRAM adresleri.*

1 kez sola kaydırma	01	02	03	04	05	06	07	08
	41	42	43	44	45	46	47	48

1 kez sağa kaydırma	27	00	01	02	03	04	05	06
	67	40	41	42	43	44	45	46

Şekil-13.8. *2x8 LCD DDRAM belleğinin 1 kez sola ve sağa kaydırma işlemi.*

Bahsedilen LCD DDRAM belleğine display'de gösterilmek istenen karakterler yerleştirilir. Peki LCD'de mevcut olan karakterler nelerdir? LCD'de standart bir karakter grubu vardır. Bu karakterler içinde standart ASCII karakterler, Yunan ve Japon harf karakterleri vardır. Karakterler LCD'de **CGROM** (Character Generator ROM) belleğinde tanımlanmışlardır. LCD karakter tablosu Tablo-13.1'de verilmiştir.

Tablo-13.1'de verilen sabit karakterler değiştirilemez. Fakat kullanıcılar en fazla 8 adet daha karakter tanımlayabilir (5x10 dot matris'e sahip LCD'lerde en fazla 4 karakter tanımlanabilir). Bu tanımlanan karakterler LCD'de **CGRAM** bellekte saklanır. **Tablo-13.1'de dikkat edilirse 0x00 ile 0x0F'e kadar olan ilk 16 karakterlik alan boştur. Bu 16 alana kullanıcılar istediği karakterleri kaydedebilir.** İstenen karakter için CGRAM'a istenen karakterin verileri gönderilmelidir. Şekil-13.9'da "R" karakteri ve veri değerleri görülmektedir. Kullanıcı istediği karakteri bu mantık çerçevesinde oluşturup, karakterin veri değerlerini çıkartmalıdır. Şekilde görüldüğü gibi 7, 6 ve 5. bit'lerin değerleri hep sıfır (0)'dır. Şekilde her dot matris satırının yanında o satırın değeri binary, heksadesimal ve desimal olarak sırayla verilmiştir. Örneğin bu örneğe bakarsak "R" karakteri için LCD'ye sırayla desimal olarak 30, 17, 17, 30, 20, 18 ve 17 sayılarının gönderilmesi gerekir.

CGRAM data								
7	6	5	4	3	2	1	0	
*	*	*	1	1	1	1	0	00011110 = 0x1E = 30
			1	0	0	0	1	00010001 = 0x11 = 17
			1	0	0	0	1	00010001 = 0x11 = 17
			1	1	1	1	0	Karakter deseni 00011110 = 0x1E = 30
			1	0	1	0	0	00010100 = 0x14 = 20
			1	0	0	1	0	00010010 = 0x12 = 18
			1	0	0	0	1	00010001 = 0x11 = 17
*	*	*	0	0	0	0	0	Kursör pozisyonu

Şekil-13.9. *R harfinin oluşturulması için CGRAM'a yazılacak veri değerleri.*

Düşük 4 Bit \ Yüksek	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)			!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)			"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)			#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)			$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)			%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)			&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)			'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)			(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)			)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)			*	:	J	Z	j	z			ェ	コ	ハ	レ	j	¥
xxxx1011	(4)			+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)			,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)			−	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)			.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)			/	?	O	_	o	←			ッ	ソ	マ	゜	ö	■

Tablo-13.1. *LCD karakter seti.*

13.1.2. HD44780 Tabanlı LCD Komutları

LCD modülü içinde bulunan kontrolcü belli komutlara göre çalışır. Bu komutlar bir mikrodenetleyici tarafından LCD'ye gönderilir. Tablo13.2'de LCD komutları görülmektedir.

Komut	Komut Kodu										Komut İcra Süresi
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Display sil	0	0	0	0	0	0	0	0	0	1	1.53 msn
İmleç Başa	0	0	0	0	0	0	0	0	1	X	1.53 msn
Giriş Modu	0	0	0	0	0	0	0	1	I/D	S	39 µsn
Display ON/OFF ve İmleç	0	0	0	0	0	0	1	D	C	B	39 µsn
İmleç/Display Kaydır	0	0	0	0	0	1	S/C	R/L	X	X	39 µsn
Fonksiyon Ayarları	0	0	0	0	1	DL	N	F	X	X	39 µsn
CGRAM Adres Seçme	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	39 µsn
DDRAM Adres Seçme	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	39 µsn
Meşgul bayrağı (BF) ve Adres sayıcı (AC) Okuma	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	0
Adrese veri yazma	1	0	D7	D6	D5	D4	D3	D2	D1	D0	43µsn
RAM'den veri okuma	1	1	D7	D6	D5	D4	D3	D2	D1	D0	43 µsn

Tablo-13.2. *HD44780 Tabanlı LCD komut seti.*

Kısaltmaların açıklamaları;

I/D = 1: Arttır (Increment) 0: Azalt (Decrement)
S = 1: Display kaydır 0: Display kaydırma
S/C = 1: Display kaydır 0: İmleç kaydır
R/L = 1: Sağa kaydır 0: Sola kaydır
D/L = 1: Veri yolu 8 bit 0: Veri yolu 4 bit
N = 1: 2 satır 0: 1 satır
F = 1: 5x10 dot matrix 0: 5x8 dot matrix
BF = 1: LCD meşgul 0: LCD meşgul değil
D = 1: Display aç 0: Display kapat
C = 1: İmleç gösterilir 0: İmleç gösterilmez
B = 1: İmleç yanıp söner 0: İmleç yanıp sönmez
AC = Adres sayıcı (Adres Counter). CGRAM ve DDRAM adreslerinin ikisi için de kullanılır
X = Önemsiz bit. "1"'de olabilir "0" da.

NOT: Her komutun işlenmesi belli bir zaman gecikmesi gerektirir. Tablo-13.2'de verilen komut icra zamanları f_{OSC}=270 Khz'e göre hesaplanmıştır. LCD'nin komut işlemesini bitirip bitirmediğini anlamanın en kesin yolu BF bit'inin kontrolünü yapmaktır.

Başlangıç Ayar Komutları:

Display Sıfırlama: Bu komut ile LCD kullanıma hazır hale getirilir ve imleç 1.satır 1.sütuna konumlanır. Bunun için LCD'ye **0x02** veya **0x03** komutu gönderilmelidir.

Display Sil: Bu komut ile display silinir ve LCD imleci 1.satır ve 1.sütuna konumlandırılır. Bunun için LCD'ye **0x01** komutu gönderilmelidir.

Fonksiyon Ayarları: Fonksiyon ayar komutları, LCD'nin satır sayısı, font büyüklüğü ve LCD ile kaç bit ile iletişim yapılacağı ile ilgili komutlarından oluşur. Tablo-13.2'ye bakarak komutların kod karşılıkları bulunabilir. Aşağıda Tablo-13.3'de örnek bazı komut ve kodları verilmiştir.

Komut	Açıklama
0x3C	LCD 1 satır, 5x8 dot matris , 4 bit iletişim
0x24	LCD 1 satır, 5x10 dot matris, 4 bit iletişim
0x28	LCD 2 satır, 5x8 dot matris , 4 bit iletişim
0x2C	LCD 2 satır, 5x10 dot matris, 4 bit iletişim
0x30	LCD 1 satır, 5x8 dot matris , 8 bit iletişim
0x34	LCD 1 satır, 5x10 dot matris, 8 bit iletişim
0x38	LCD 2 satır, 5x8 dot matris , 8 bit iletişim
0x20	LCD 2 satır, 5x8 dot matris , 8 bit iletişim

Tablo-13.3. *Fonksiyon ayarları komutları.*

Giriş Modu Komutları: LCD'de yapılan işlemler sonucunda display ve imlecin ne yapması gerektiğini belirten komutlardır. Tablo-13.2'de giriş modu kısmına bakılırsa aşağıda Tablo-13.4'de verilen komutlar görülebilir.

Komut	Açıklama
0x04	LCD'de gösterilen her karakterden sonra imleç bir sola kayar.
0x05	LCD'de gösterilen her karakterden sonra imleç bir sola kayar ve display'de bir sola kayar.
0x06	LCD'de gösterilen her karakterden sonra imleç bir sağa kayar.
0x07	LCD'de gösterilen her karakterden sonra imleç bir sağa kayar ve display'de bir sağa kayar.

Tablo-13.4. *Giriş modu komutları.*

Display ve İmleç Komutları:

Display ON/OFF ve İmleç Komutları: Display'in açılıp kapatılması, imleç çizgisinin görünüp görünmemesi veya yanıp sönmesi gibi ayarların yapılabildiği komutlardır. Tablo-13.2'de Display On/Off ve İmleç kısmına bakılırsa Tablo-13.5'de verilen komutlar görülebilir.

Komut	Açıklama
0x08	Display kapalı. İmleç alt çizgi gösterilmez.. İmleç yanıp sönmez.
0x09	Display kapalı. İmleç alt çizgi gösterilmez. İmleç yanıp söner.
0x0A	Display kapalı. İmleç alt çizgi gösterilir. İmleç yanıp sönmez.
0x0B	Display kapalı. İmleç alt çizgi gösterilir. İmleç yanıp söner.
0x0C	Display açık. İmleç alt çizgi gösterilmez. İmleç yanıp sönmez.
0x0D	Display açık. İmleç alt çizgi gösterilmez. İmleç yanıp söner.
0x0E	Display açık. İmleç alt çizgi gösterilir. İmleç yanıp sönmez.
0x0F	Display açık. İmleç alt çizgi gösterilir. İmleç yanıp söner.

Şekil-13.5. *Display On/Off ve imleç komutları.*

İmleç/Display Kaydırma Komutları: İmlecin sola-sağa kaydırılması ve display'in tümüyle sola-sağa kaydırılması işlemlerini sağlayan komutlardır. Tablo-13.2'de İmleç/Display Kaydır kısmına bakılırsa Tablo-13.6'da verilen komutlar görülebilir.

Komut	Açıklama
0x10	İmleci sola kaydır.
0x14	İmleci sağa kaydır.
0x18	Display'i sola kaydır.
0x1C	Display'i sağa kaydır.

Şekil-13.6. *İmleç/Display komutları.*

DDRAM Adres Seçme Komutu (İmleç Satır ve Sütun Konumlandırmak):

Bu komut ile DDRAM adresi belirlenir. Böylece LCD'ye gönderilen karakterler DDRAM adresi ile belirlenen satır ve sütun'dan başlayarak yazılır. Önceden verilen Şekil-13.7'ye bakarsanız 2x16 bir LCD'de 1.satır 1. sütun DDRAM adresi 0x00 iken, 2.satır 1.sütun adresi 0x40'dır. İstenen satır ve sütun adresi için satır başı adres değerine istenen sütun değerinin bir eksiği eklenir ve DDRAM adresi bulunur. Örneğin 2.satır ve 5.sütun adresi için, 2.satır başlangıç adresi olan 0x40 sayısına sütun değeri 5'in 1 eksiği olan 4 sayısı eklenir. Sonuçta 2.satır 5.sütun DDRAM adres değeri 0x44 olarak bulunur. Tablo-13.2'de verilen DDRAM komutuna tekrar bakalım. Bu komuttaki AC6...AC0 bit'leri DDRAM adresini belirten bit'lerdir. İlk bit hep sabittir.

1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

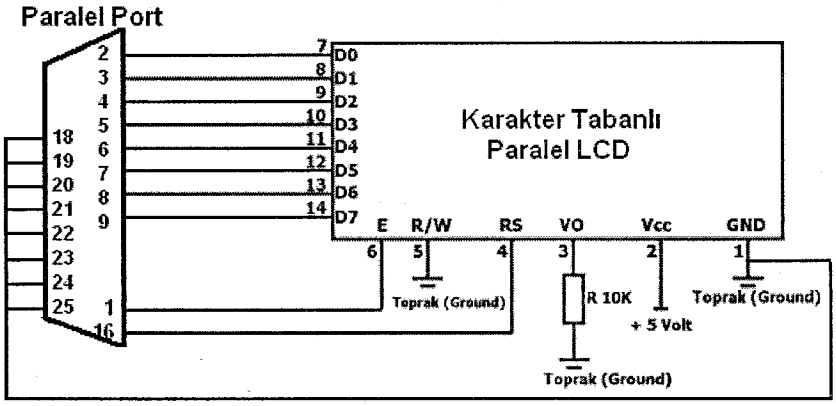
Örneğin LCD'de 2.satır 5. sütundan itibaren yazı yazdırmak istediğimizi farz edelim. Bunun için ilk başta LCD'nin istenen satır ve sütundaki DDRAM adresine bakılır (Şekil-13.7). İstenen DDRAM adresinin heksadesimal 0x44 olduğu görülür. Şimdi komutumuzun kod değerini bu DDRAM adresini yazarak bulalım. Sonuç olarak elde

edilen komut değeri 0xC4 bulunur. **Bu mantıkla imleç istenen satır ve sütuna konumlandırılabilir.**

1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	
1	1	0	0	0	1	0	0	= 0xC4
	4			4				

13.1.3. Paralel LCD'yi Direkt Bilgisayara Bağlamak

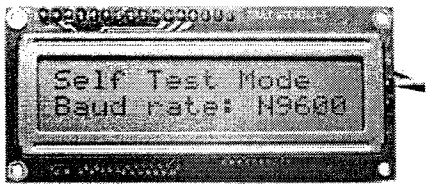
Paralel LCD'leri direkt olarak bilgisayarın paralel port'una bağlayabiliriz. Böylece çoğu uygulamaları denetleyici kullanmadan doğrudan bilgisayardan kontrol edebiliriz. Örneğin bilgisayar işlemcisinin sıcaklığını, fan hızını vb. bilgileri LCD'de gösteren bir program yaptığımızı düşünelim. Bu program için hiç ek devreye gerek duymadan, direkt olarak paralel LCD, bilgisayarın paralel port'una bağlanır ve uygulama gerçekleştirilmiş olur. Bilgisayarın paralel port'una paralel LCD bağlantısı için Şekil-13.10'da verilen bağlantı kullanılabilir.



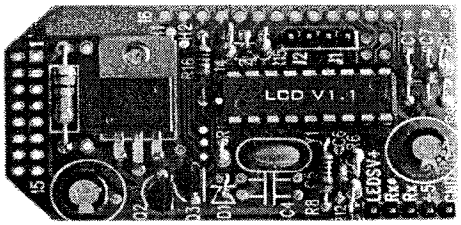
Şekil-13.10. *Paralel port'a karakter tabanlı paralel LCD bağlantısı.*

13.2. SERİ LCD

Seri LCD'ler görünüm olarak Paralel LCD'lere benzerler. Paralel LCD'lerde 4 bit veri iletimi modun da toplam 7 pin kullanılmak zorundadır. Okuma yapılmayacağı durumlarda da (R/W ucu şaseye bağlanır) en azından 6 pin'e ihtiyaç vardır. Fakat seri LCD'de veri iletimi için sadece bir pin'e ihtiyaç vardır. Seri LCD'de iletim adı üstünde seri iletişim ile yapılır. Şekil-13.11'de seri karakter LCD, Şekil-13.12'de de herhangi bir seri LCD'nin arka yüzü görülmektedir.

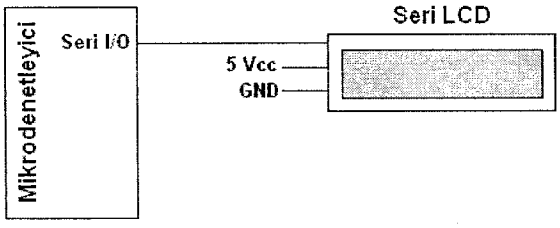


Şekil-13.11. *Seri karakter LCD*



Şekil-13.12. *Seri karakter LCD arka yüzü*

Seri LCD'nin mikrodenetleyiciye bağlantısı için tek bir seri giriş çıkış ucu (seri I/O) ve pozitif besleme (+ 5 Vdc) ile şase (GND) bağlantısı yeterlidir. Seri LCD'lerin arka yüzünde seri iletişim hızı ayar anahtarı veya jumper bağlantıları vardır. Seri iletişim hızı 2400, 9600, 19200 veya 57600 olarak seçilebilen seri LCD'ler mevcuttur. Seri LCD'ler de aynı zamanda paralel LCD'ler gibi paralel veri girişleri de mevcuttur.



Şekil-13.13. *Seri LCD bağlantı uçları.*

13.2.1. Seri LCD Komutları ve DDRAM Adresleri

Seri LCD'lere komut göndermek için komut göndermeden önce heksadesimal 0xFE veya desimal 254 bilgisi LCD'ye gönderilmeli ardından komut kodu gönderilmelidir. Seri LCD komutları Tablo-13.7'de verilmiştir.

Komut	Komut Kodu	
	Hex	Desimal
Display sil	0x01	1
İmleç sol satır başına	0x02	2
İmleci gizle	0x0C	12
İmleci bir sola kaydır	0x10	16
İmleci bir sağa kaydır	0x14	20
İmleç yanıp sönsün	0x0D	13
İmleç alt çizgi modunda	0x0E	14
İmleci ikinci satıra indir	0xC0	192
İmleci üçüncü satıra indir	0x94	148
İmleci dördüncü satıra indir	0xD4	212

Tablo-13.7. *Seri LCD Komutları.*

Seri LCD komut tablosunda görüldüğü gibi seri LCD'lerin DDRAM adresleri 0x80 değerinden başlar. Şekil-13.14'te 16x2 ve Şekil-13.15'de de 20x4 seri LCD DDRAM adresleri görülmektedir.

80	81	82	83	84	85	85	87	88	89	8a	8b	8c	8d	8e	8f
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	ca	cb	cc	cd	ce	cf

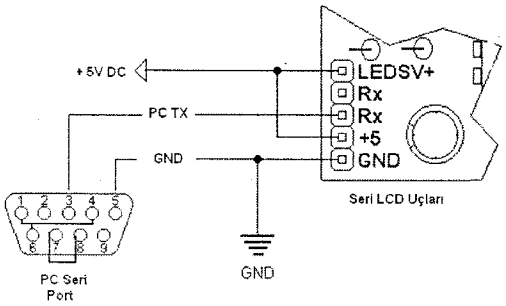
Şekil-13.14. *16x2 seri LCD DDRAM adresleri.*

80	81	82	83	84	85	85	87	88	89	8a	8b	8c	8d	8e	8f	90	91	92	93
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	ca	cb	cc	cd	ce	cf	d0	d1	d2	d4
94	95	96	97	98	99	9a	9b	9c	9d	9e	9f	a0	a1	a2	a3	a4	a5	a6	a7
d4	d5	d6	d7	d8	d9	da	db	dc	dd	de	df	e0	e1	e2	e3	e4	e5	e6	e7

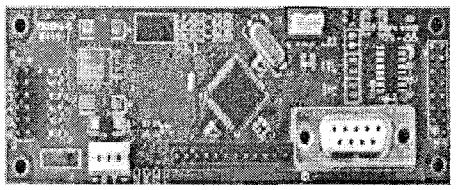
Şekil-13.15. *20x4 seri LCD DDRAM adresleri.*

13.2.2. Seri LCD'yi Direkt Bilgisayara Bağlamak

Seri LCD'leri direkt olarak bilgisayarın seri port'una bağlayabiliriz. Böylece çoğu uygulamaları denetleyici kullanmadan doğrudan bilgisayardan kontrol edebiliriz. Şekil-13.16'da bu bağlantı görülmektedir. Seri LCD'lerin arka yüzlerinde seri port bulunan modelleri de vardır (Şekil-13.17).



Şekil-13.16. *Seri LCD'yi Bilgisayarın seri port'una bağlamak.*



Şekil-13.17. *Seri konnektörlü seri LCD*

13.3. CCS C KARAKTER TABANLI PARALEL LCD KÜTÜPHANESİ (LCD.c ve LCD420.c)

CCS C derleyicisi paralel LCD'lerle iletişim için kütüphanesinde **"LCD.c"** dosyasını içerir. Bu dosya sayesinde karakter tabanlı paralel LCD'ler ile kolayca iletişim kurulabilmektedir. Bu dosyayı kullanarak yukarıda LCD'ler için anlatılan bilgileri bilmeden kolayca LCD'ye yazı yazdırılabilir. "LCD.c" dosyası 2 satır LCD'ler içindir. **"LCD420.c"** dosyası ise 4 satırlık 20 sütunluk (4x20) LCD'ler içindir. İki dosya komutları da aynıdır. Bunun gibi sürücü dosyaları CCS programınızı kurduğunuz dizinde bulunurler. Örneğin CCS C'yi standart olarak tanımlı bir dizine kurduysanız bu tür dosyalar, **C:\Program Files\PICC\Drivers** klasörü içindedirler. Bu dosyaları kullanmak için programın başında kullanılacak LCD dosyası `#include <lcd.c>` komutu veya `#include <lcd420.c>` ile programa eklenmelidir. Bu dosyalar, LCD'nin D veya B port'una bağlanması ve aşağıdaki bağlantıların yapılmasını gerektirir.

LCD D port'una bağlanırsa;	**LCD D port'una bağlanırsa;**
D0 enable	B0 enable
D1 RS	B1 RS
D2 R/W	B2 R/W
D4 D4	B4 D4
D5 D5	B5 D5
D6 D6	B6 D6
D7 D7	B7 D7

LCD.c ve LCD420.c dosyalarında LCD kontrolü için hazır fonksiyonlar bulunmaktadır. Bu fonksiyonları ve anlamları aşağıda açıklanmıştır.

`lcd_init ()` = Diğer LCD fonksiyonları çağrılmadan bu fonksiyon mutlaka bir kez çağrılmalı.

`lcd_putc(c)` = LCD'ye karakter veya string yazmak için kullanılır. Parantez içine gönderilecek karakter veya string yazılır. Tek karakter gönderilirken tek tırnak içine alınır 's'. Eğer string ifade gönderilecekse çift tırnak içine alınır "CCS". Aynı zamanda parantez içine aşağıda verilen ters eğik çizgi sabitleri de getirilebilir
\f = Display'i sil.
\n = Satır atla.
\b = Bir karakter geri gel.

`lcd_gotoxy(x,y)` = İmlecin yerini belirlemeye yarayan fonksiyondur. Parantez içindeki x yerine sütun değeri, y yerine satır değeri yazılır.

`lcd_getc(x,y)` = O an ki imlecin belirtilen konumunda bulunan karakter ile geri dönen bir fonksiyondur.

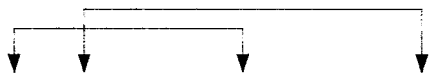
`#define use_portb_lcd TRUE` = Eğer LCD, D port'una değil de B port'una bağlanacaksa bu komutun `#include <lcd.c>` komutundan önce programda yer alması gerekmektedir.

`lcd_send_byte(byte adres,byte bilgi)` = LCD'ye istenen bir komutu gönderir.

`lcd_read_byte()` = LCD'den 1 byte veri okur.

`printf(lcd_putc,"...");` = lcd_putc() komutu ile LCD'ye karakter veya string yazdırmak yerine printf(lcd_putc,"...") komutu kullanılabilir. printf komutu C dilinde olan bir komuttur. CCS C'de tüm özellikleri ile bu komuta uyumludur. Bu komut kullanılarak LCD'ye değişken, sabit veya string belli bir format ile gönderilir. Uygulamalarda bu komutu daha iyi anlayabilirsiniz. **Kitapta programlarımızda LCD işlemleri için hep printf(lcd_putc,"...") komutu kullanılacaktır.**

Printf komutunda LCD için parantez içine ilk başta kullanılacak LCD komutu yazılır sonra virgül (,) konulur ve çift tırnak açılarak gerekli bilgiler ve sabitler yazılır. Aşağıda görüldüğü gibi açılan çift tırnaktan sonra lcd_putc() komutunda olduğu gibi ters eğik çizgi kodları **(\f)** kullanılabilir. Komutta **%** işareti ve ondan sonra gelen kod, değişkenin ekranda hangi formatta yazılacağını belirtir. % sabitleri için Bölüm 4.6.2'de Tablo-4.4'e bakabilirsiniz. Örneğin aşağıdaki komutta %d ile belirtilen yere tam sayı tipinde (d kodu bunu belirtir) "değişken1" adlı değişkenin içeriği yazılır. Hemen bu ifadenin yanına da %f ile belirtilen yere ondalıklı tipte, "değişken2" adlı değişkenin içeriği yazılır.



printf(lcd_putc," \f\n\b *<ifade>* %<d>%<f>...", <değişken1>,<değişken2>,...);

```
printf(lcd_putc,"\f \nSicaklik=%d C",temp);
```

Yukarıdaki örnekte ilk başta \f komutu ile LCD silinmektedir. Ardından \n komutu ile de LCD'ye satır atla komutu verilmiştir. Böylece LCD imleci ikinci satıra geçmiştir. LCD'ye "Sicaklik=" yazısı yazdırılmış ardından da, %d komutunun olduğu yere "temp" değişkeninin içeriği tam sayı formatında yazdırılmıştır. En son olarakda "C" harfi LCD'ye yazdırılmıştır. Örneğin "temp" değişkeninin içeriğinin 30 olduğunu farz edersek LCD'de aşağıdaki görüntü oluşur.

```
Sicaklik=30 C
```

NOT: lcd.c dosyası içinde gecikme fonksiyonları kullanıldığından. Programda gecikme fonksiyonlarının kullanacağı osilatör frekansını belirten

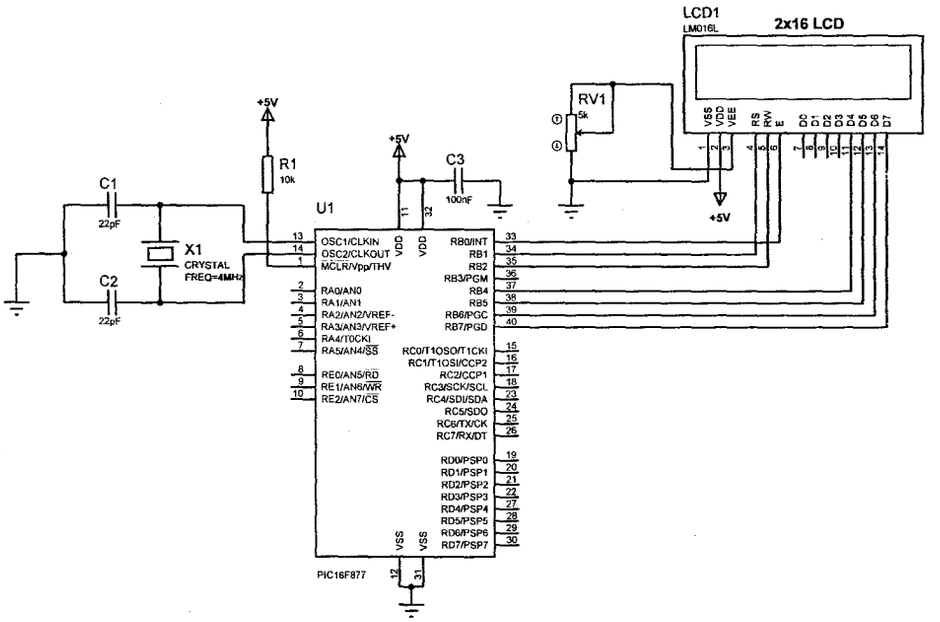
`#use delay (4000000)` komutu mutlaka `#include <lcd.c>` komutundan önce belirtilmelidir.

NOT: İnternette değişik kaynaklarda veya CCS C forum sayfalarında (http://www.ccsinfo.com/forum) başka LCD çeşitleri için ve LCD'nin istenen pin'lerine bağlanabilen şekilde yazılmış LCD rutinlerini içeren dosyalar rahatlıkla bulunabilir. Böylece kullandığınız LCD için hazır fonksiyonlar elde etmiş olursunuz.

13.4. PARALEL LCD UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulama da CCS C programının hazır LCD dosyası ile bir uygulama gerçekleştirilmiştir. Uygulamada 2x16 LCD kullanılmıştır. LCD B port'una bağlanmıştır. LCD'de çeşitli stringler gösterilerek, LCD'ye komut gönderilerek, LCD'de imleç konumlandırması gibi tüm işlemler bu uygulamada yapılarak LCD kullanımına örnek bir uygulama gerçekleştirilmiştir.



Şekil-13.18. *Paralel LCD uygulaması devre şeması.*

PIC PROG/DEKA 16F87x deneme kartında LCD için iki taraflı bir soket bulunmaktadır. **Deneme kartının LCD bağlantısı CCS C programının hazır LCD dosyasındaki bağlantıya uymamaktadır.** Bu nedenle LCD uygulamasını denemek için LCD'nizi bir board üzerine yerleştirin ve deneme kartının sağ

tarafında LED grubunun altında bulunan B port'u çıkış pin'lerinden zil teli kablosu ile aldığınız RB0, RB1, RB2,...RB7 uçlarını board üzerinde bulunan LCD'nize bağlayınız. LCD'nin Vdd ve Vss besleme bağlantılarını da deneme kartında bulunan LCD soketinden alabilirsiniz. Port B jumper'ının LCD konumunda olması gerektiğini unutmayınız.

Devre programı;

```
/*******************************************************************
      PIC16F877 ile Paralel LCD Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*******************************************************************/

#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.

#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

#define use_portb_lcd TRUE   // LCD B port'una bağlı

#include <lcd.c>   // lcd.c dosyası tanıtılıyor

int x;       // Tamsayı tipinde değişken tanımlanıyor
float y;     // Ondalıklı tipte değişken tanımlanıyor
char z;      //  Karakter tipinde değişken tanımlanıyor

//********** ANA PROGRAM FONKSİYONU*******
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x00);   // B port'u komple çıkış

   lcd_init();   // LCD hazırlanıyor

   x=10;   // x değişkenine değer veriliyor
   y=32.6; // y değişkenine değer veriliyor
```

```
   while(1) // Sonsuz döngü
   {
      lcd_send_byte(0,0x0d); // LCD'ye komut gönderiliyor. İmleç
                             // yanıp sönsün komutu.
      printf(lcd_putc,"\f     CCS C"); // LCD'ye veri gönderiliyor.
      delay_ms(1000); // 1 sn gecikme
      printf(lcd_putc,"\n  Derleyicisi");//LCD'ye veri gönderiliyor.
      delay_ms(1000); // 1 sn gecikme
      lcd_gotoxy(1,2);// Kursör 1.sütun, 2.satıra konumlandırılıyor.
      printf(lcd_putc,"\fX DEGERI=%d",x); // LCD'de tamsayı olarak x
                                  // değişken değeri görüntüleniyor.
      delay_ms(1000); // 1 sn gecikme
      printf(lcd_putc,"\f\nY DEGERI=%f",y), // LCD'de ondalıklı
                  // olarak y değişken değeri görüntüleniyor.
      delay_ms(1000); // 1 sn gecikme
      printf(lcd_putc,"\fDIRENC"); // LCD siliniyor ve "Direnc"
                                   // stringi yazdırılıyor.
      delay_ms(1000); // 1 sn gecikme
      z=lcd_getc(3,1);   // LCD'de o an 3.Sütun ve 1.Satırda olan
       //karakter alınıyor. Alınan değer z değişkenine aktarılıyor.
      printf(lcd_putc,"\f%c",z); // z değişkeni LCD'ye aktarılıyor
      delay_ms(2000); // 2 sn gecikme
   }
}
//*****************************************************************
```

Programda `#define use_portb_lcd TRUE` komutu ile LCD'nin B port'una bağlı olduğu derleyiciye bildirilmiştir. Eğer LCD D port'una bağlı olsaydı bu komutu ve bunun yerine başka bir komut kullanmayacaktır. `#include <lcd.c>` komutu ile de LCD.c dosyası programa dahil edilmiştir. `int x;`, `float y;` ve `char z;` komutları ile değişkenler tanımlanmıştır.

Ana fonksiyonda önceden belirtildiği gibi `lcd_init();` komutu kullanılmıştır. Tüm LCD dosyası komutları kullanılmadan önce bir kez mutlaka bu komut kullanılmış olmalıdır. `x=10;` ve `y=32.6;` komutları ile değişkenlere değer yüklenmiştir.

`while(1)` ile oluşturulan döngüde `lcd_send_byte(0,0x0d);` komutu ile LCD'ye komut gönderilmiştir. 0x0d komutu imlecin yanıp sönmesini sağlayan komuttur. Ardından `printf(lcd_putc,"\f  CCS C");` komutu ile de ilk başta LCD temizlenmiş (komuttaki \f kısmı ile) ve LCD'ye "CCS C" stringi (ifadesi) gönderilmiştir. LCD'ye her gönderilen değer ardından belli bir gecikme verilmiştir. `printf(lcd_putc,"\n  Derleyicisi");` komutu ile de LCD'de ilk başta bir alt satıra geçilmiş (komuttaki \n kısmı ile) ve LCD'ye "Derleyicisi" stringi yazdırılmıştır. `lcd_gotoxy(1,2);` komutuyla imleç 1.sütun, 2.satıra konumlandırılmıştır.

`printf(lcd_putc,"\fX DEGERI=%d",x);` komutu ile ilk başta LCD temizlenmiş (\f kısmı ile) ardından "X DEGERI=" ifadesi LCD'ye yazdırılmıştır. Komuttaki `"%d"` kısmı ile buraya değişken değeri yazılacağını ve bu değişken değerinin tam sayı formatında yazdırılacağı belirtilmiştir (Kitapta 4.6.2. bölümünde % sabitleri kısmına göz atabilirsiniz). Komutta çift tırnaktan sonra virgül (,) konulmuş ve virgülden

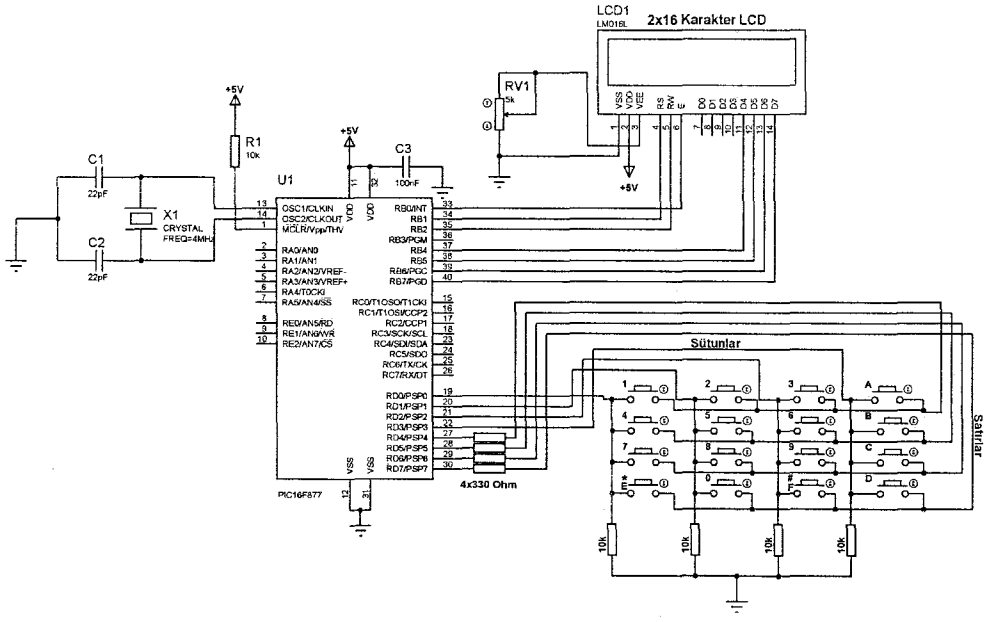
sonra ise komuttaki %d kısmına yazılacak değişkenin ismi yazılmıştır. Sonuçta bu komutla x değişkeni değeri ekrana `"x DEGERI="` ifadesinden hemen sonra yazdırılacaktır. Programdaki diğer benzer komutlarda bu mantıkla kullanılmıştır.

Programdaki `z=lcd_getc(3,1);` komutu ile de LCD'de 3. sütun ve 1.satırda o an bulunan karakter alınıp z değişkenine aktarılmıştır. Daha sonra ise `printf(lcd_putc,"\f%c",z);` komutu ile bu z değişkeni LCD'ye yazdırılıyor. Komutta dikkat edilirse %c ifadesi ile LCD'ye yazdırılan z değişkeni değerinin karakter formatında LCD'ye yazdırılması sağlanmıştır.

13.5. TUŞ TAKIMI VE LCD UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulama aslında önceki bölümde incelenen Tuş Takımı-1 uygulaması ile bu bölümdeki paralel LCD uygulamasının birleştirilmiş halidir. Uygulama ile Tuş takımından basılan tuşun değeri LCD'de gösterilmesi sağlanmıştır.



Şekil-13.19. *Tuş takımı ve LCD uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Tuş Takımı ve LCD Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
```

```
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli
#use fast_io(d) //Port yönlendirme komutları B port'u için geçerli

#define use_portb_lcd TRUE   // LCD B port'una bağlı

#include <lcd.c>   // lcd.c dosyası tanıtılıyor
#define sut1   pin_d0 // sut1 ifadesi pin_d0 ifadesine eşitleniyor
#define sut2   pin_d1 // sut2 ifadesi pin_d1 ifadesine eşitleniyor
#define sut3   pin_d2 // sut3 ifadesi pin_d2 ifadesine eşitleniyor
#define sut4   pin_d3 // sut3 ifadesi pin_d2 ifadesine eşitleniyor

#define sat1   pin_d4 // sat1 ifadesi pin_d4 ifadesine eşitleniyor
#define sat2   pin_d5 // sat2 ifadesi pin_d5 ifadesine eşitleniyor
#define sat3   pin_d6 // sat3 ifadesi pin_d6 ifadesine eşitleniyor
#define sat4   pin_d7 // sat4 ifadesi pin_d7 ifadesine eşitleniyor

char tus=0; // karakter tipinde değişken tanımlanıyor

//*********** Keypad Tarama Fonksiyonu *********
char keypad_oku() // Fonksiyon ismi
{
   output_d(0x00);     // D port'u çıkışı sıfırlanıyor

   output_high(sat1); // 1. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=1; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=2; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=3; }
   if (input(sut4))   // 4. sütun okunuyor
      { delay_ms(20); tus=0xA; }
   output_low(sat1); // 1. satır lojik-0 yapılıyor

   output_high(sat2); // 2. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=4; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=5; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=6; }
   if (input(sut4))   // 4. sütun okunuyor
      { delay_ms(20); tus=0xB; }
   output_low(sat2); // 2. satır lojik-0 yapılıyor
```

```
   output_high(sat3); // 3. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=7; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=8; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=9; }
   if (input(sut4))   // 4. sütun okunuyor
      {delay_ms(20); tus=0x0C; }
   output_low(sat3); // 3. satır lojik-0 yapılıyor

   output_high(sat4); // 3. satır lojik-1 yapılıyor
   if (input(sut1))   // 1. sütun okunuyor
      { delay_ms(20); tus=0xE; }
   if (input(sut2))   // 2. sütun okunuyor
      { delay_ms(20); tus=0; }
   if (input(sut3))   // 3. sütun okunuyor
      { delay_ms(20); tus=0xF; }
   if (input(sut4))   // 4. sütun okunuyor
      {delay_ms(20); tus=0xD; }
   output_low(sat4); // 3. satır lojik-0 yapılıyor

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

//********** ANA PROGRAM FONKSİYONU*******
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_b(0x01);  // B port'u komple çıkış
   set_tris_d(0x0F);  // Yüksek değerlikli 4 bit çıkış, düşük
                      // değerlikli 4 bit giriş

   lcd_init();   // LCD hazırlanıyor

   printf(lcd_putc,"\fBasilan Tus="); // LCD'ye string yazdırılıyor

   while(1) // Sonsuz döngü
   {
      lcd_gotoxy(13,1); // İmleç 3.sütun, 1.satıra konumlandırılıyor

      if(keypad_oku()!='\0') // Tuş takımında bir tuşa basılmışsa
       {
        if (keypad_oku()>9)//Eğer basılan tuş değeri 9'dan büyük ise
```

```
            printf(lcd_putc,"%d",keypad_oku());    //Tuş değeri LCD'ye
                                                   // yazdırılıyor
        else  // Eğer basılan tuş değeri 9'dan büyük değilse
            printf(lcd_putc,"%d ",keypad_oku()); //Tuş değeri LCD'ye
                                                   // yazdırılıyor
        }
}
//*****************************************************************
```

Programda kullanılan komutların çoğu önceden tuş takımı uygulaması-1 ve paralel lcd uygulaması örneklerinde anlatılmıştır. Bu komutlardan farklı olarak programda `printf(lcd_putc,"\fBasilan Tus=");` komutu ile "Basilan Tus=" yazısı LCD'ye yazdırılmaktadır. Bu yazı 12 karakterdir. Programda sonsuz döngüde `lcd_gotoxy(13,1);` komutu ile imleç her zaman 13.sütun ve 1.satırda tutulmaktadır. Böylece LCD'de "Basilan Tus=" yazısı her zaman sabit kalacaktır. Basılan tuş değerleri ise 13.sütundan başlayarak LCD'de görünecektir. Örneğin LCD ekranda "Basilan Tus=13" yazısı görüldükten sonra tuş takımından "5" tuşuna basıldığında ekranda "Basilan Tus=53" yazısı görülecektir. Bunu önlemek için `if (keypad_oku()>9)` komutu ile eğer basılan tuş değeri 9'dan büyük ise LCD'ye tuş değeri normal şekilde 13.sütundan başlanarak yazdırılmıştır. Eğer basılan tuş değeri 9'dan büyük değilse else komutunun altındaki komut işlem görecektir. Bu komuta dikkat edilirse `printf(lcd_putc,"%d ",keypad_oku());`, komutta `%d` yerine `keypad_oku()` fonksiyonunun geri dönüş değeri yazdırılacaktır. Fakat aynı zamanda komutta, "%d " kısmına bakılırsa fonksiyonunun geri dönüş değeri yazdırıldıktan sonra bir karakterlik boşluk da verilmiştir. Böylece bir önceki değer örneğin "Basilan Tus=13" iken yeni basılan tuş değeri "5" olduğunda LCD'de "Basilan Tus=5 " yazısı görülecektir. if-else döngüsü ve verilen 1 karakterlik boşluk sayesinde yukarıda anlatılan sorun çözülmüş olmaktadır.

13.6. KENDİ LCD DOSYAMIZI OLUŞTURMAK

CCS C programı kendi içinde LCD dosyaları barındırmaktadır. Aynı zamanda CCS C resmi forum sitesinde ve diğer internet kaynaklarında da CCS C için yazılmış çeşitli LCD sürücü dosyaları bulunabilir. Fakat sizin kullandığınız LCD farklı yapıda ve özellikte olabilir yada LCD'nizi farklı pin'lere bağlamak isteyebilirsiniz. Bu gibi durumlarda kendi LCD dosyanızı oluşturabilmek büyük avantaj ve esneklik getirir. Aynı zamanda da LCD kontrolü daha iyi kavranmış olunur. Bu kısımda 2x16'lık bir LCD için bir LCD fonksiyonları içeren bir dosya oluşturulmuştur. Bu dosyaya da **"benim_LCD.c"** adı verilmiştir. LCD'den veri okuma yapılmayacağı için R/W ucu şaseye direkt bağlanmıştır.

"benim_LCD.c" Dosyası İçeriği:

```
//////////////////////////////////////////////////////////////
//                    benim_LCD.c                           //
// Bu LCD sürücü dosyası 2x16 HD44780 uyumlu LCD'ler        //
// için yazılmıştır. LCD'den sadece okuma yapılacağı için   //
// R/W ucu şaseye bağlanmalıdır. LCD bağlantısı aşağıdadır. //
//////////////////////////////////////////////////////////////
// RB0 D4
// RB1 D5
// RB2 D6
// RB3 D7
// RB4 RS
// RB5 E
// R/W ucu direkt şaseye bağlanacak

#define e   pin_b5 // LCD'nin E ucu RB5 pin'ine bağlı
#define rs  pin_b4 // LCD'nin RS ucu RB4 pin'ine bağlı

//****** LCD'ye Komut Gönderme Fonksiyonu **********
void lcd_komut(byte komut)
{
   output_b(komut>>4);  // Komutun yüksek değerli 4 bit'ini gönder
   output_low(rs);      // LCD komut almak için ayarlandı
   output_high(e);      // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(2);         // 2 msn gecikme veriliyor

   output_b(komut&0x0F); // Komutun düşük değerli 4 bit'ini gönder
   output_low(rs);       // LCD komut almak için ayarlandı
   output_high(e);       // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(2);          // 2 msn gecikme veriliyor
}

//******* LCD'ye Veri Gönderme Fonksiyonu **********
void lcd_veri(byte veri)
{
   output_b(veri>>4);   // Verinin yüksek değerli 4 bit'ini gönder
   output_high(rs);     // LCD veri almak için ayarlandı
   output_high(e);      // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(1);         // 1 msn gecikme veriliyor

   output_b(veri&0x0F); // Verinin düşük değerli 4 bit'ini gönder
   output_high(rs);     // LCD veri almak için ayarlandı
   output_high(e);      // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(1);         // 1 msn gecikme veriliyor
}

//******* LCD'de İmlec Konumlandırma Fonksiyonu ********
```

```
void imlec(byte satir, byte sutun)
{
   if (satir==1)          // Eğer satır değişkeni "1" ise
      lcd_komut(0x80|(sutun-1));

   if (satir==2)          // Eğer satır değişkeni "2" ise
      lcd_komut(0x80|(0x40+(sutun-1)));
}

//********* LCD Başlangıç Ayarları Fonksiyonu ******
void lcd_hazirla()
{
   lcd_komut(0x02);// LCD'yi kullanıma hazır hale getir, imleç
                   // 1.satır 1.sütunda komutu
   lcd_komut(0x28);//2 satır, 4 bit iletişim, 5x8 dot matris seçildi
   lcd_komut(0x08);//Display kapalı,alt çizgi ve yanıp sönme yok
   lcd_komut(0x0C);//Display açık,imleç alt çizgi ve yanıp sönme yok
   lcd_komut(0x06);// Her veri yazıldığında imleç bir sağa gitsin
   lcd_komut(0x01);// Display sil. İmleç 1.satır 1.sütunda
   delay_ms(1);    // 1 msn bekle
}
//*****************************************************************
```

Dosyada ilk başta `#define e pin_b5` ve `#define rs pin_b4` komutları ile e ve rs ifadeleri ilgili pin isimlerine eşitlenmiştir. Ardından `void lcd_komut(byte komut)` ile LCD'ye bir byte'lık komut gönderme fonksiyonu tanımlanmıştır. LCD ile 4 bit üzerinden haberleşme yapılacağından 8 bit'lik (1 byte) komut bilgisinin ilk önce yüksek değerlikli 4 bit'i, ardından da düşük değerlikli 4 bit'i gönderilmelidir. Bunun için `output_b(komut>>4);` komutu ile komut bilgisi 4 kez sağa kaydırılmıştır. Böylece 8 bit'lik komut bilgisinin yüksek değerlikli 4 bit'i ilk 4 bit'e denk gelmiştir. Yani komut değeri 0000XXXX şeklini almıştır ve bu değer B port'una gönderilmiştir. Sonuçta RB0, RB1, RB2 ve RB3 pin'lerinden komut değerinin yüksek değerlikli 4 bit'i gönderilmiş olmaktadır. Önceden anlatıldığı gibi LCD'nin veri yolundaki bilgiyi komut olarak değerlendirmesi için RS ucunun "0" yapılması gereklidir. Bunun için `output_low(rs);` komutu ile RS ucu lojik-0 yapılmıştır. LCD'nin istenen bir işlemi yapabilmesi için bilindiği gibi E ucunun lojik-1'den lojik-0'a çekilmesi gerekmektedir. `output_high(e);` ve `output_low(e);` komutları ile bu işlem yapılmıştır. Ardından işlemlerin yapılabilmesi için `delay_ms(2);` komutu ile 2 msn'lik gecikme verilmiştir. `output_b(komut&0x0F);` komutu ile de komut bilgisinin düşük değerlikli 4 bit'i B port'una gönderilmiştir. Bu işlem için komut bilgisi 0x0F sayısı ile çarpılmıştır (maskeleme olayı). Böylece yüksek değerlikli 4 bit sıfırlanmış, sadece düşük değerlikli 4 bit kalmıştır. Diğer işlemler ile de komut bilgisinin LCD'ye gönderilmesi için yüksek değerlikli 4 bit'in gönderilmesinde olduğu gibi aynen uygulanmıştır. Her gönderimde LCD'nin veri yolundaki bilginin veri olarak algılamasını sağlamak için RS ucu `output_high(rs);` komutu ile lojik-1 yapılmıştır. Her işlemin gerçekleşmesi için ise `output_high(e);` ve `output_low(e);` komutları ile E pin'i lojik-1'den lojik-0'a çekilmiştir.

`void lcd_veri(byte veri)` komutu ile LCD'ye bir byte'lık veri gönderme fonksiyonu oluşturulmuştur. Bu fonksiyonda da LCD ile 4 bit'lik iletişim kurulduğundan, 8 bit'lik veri bilgisinin ilk başta yüksek değerlikli 4 bit'i, ardından da düşük değerlikli 4 bit'i gönderilmiştir. `output_b(veri>>4);` komutu ile verinin yüksek değerlikli 4 bit'i, `output_b(veri&0x0F);` komutu ile de verinin düşük değerlikli 4 bit'i gönderilmiştir. Verinin her 4'lü grubunun gönderiminden sonra `delay_ms(1)` komutu ile 1 msn'lik gecikme verilmiştir.

LCD imlecinin istenen yere konumlandırılması işlemleri içinde `void imlec(byte satir, byte sutun)` komutu ile imleç konumlandırma fonksiyonu tanımlanmıştır. Önceden anlatıldığı üzere imleci konumlandırmak için DDRAM adres seçme komutu kullanılır. DDRAM adres seçme kısmına tekrar göz atmanız tavsiye edilir. DDRAM adres seçme komutu **1 AC6 AC5 AC4 AC3 AC2 AC1 AC0** bit'lerinden oluşur. AC bit'leri DDRAM adres değeridir. En baştaki 1 ise hep sabittir. 2x16'lık bir LCD'nin ilk satır adresi 0x00'dan başlar. Bu değere istenen sütun değerinin bir eksiği eklenerek istenen DDRAM adresi elde edilir. DDRAM adresi belirleme komut değerini bulmak için ise elde edilen adres değerinin başına "1" bit'i gelmelidir. Bu nedenle ilgili DDRAM adres değeri 0x80 sayısı ile toplanarak (OR işlemi) adres değerinin başına "1" bit'i getirilmiş olur ve elde edilen sonuç DDRAM adresini belirleme komut değeri olur. Bu nedenle `if (satir==1)` komutu ile satır değişkeni eğer "1" ise, `lcd_komut(0x80|(sutun-1));` komutunun işlem görmesi sağlanmıştır. Bu komutla sutun değerinin 1 eksiği 0x80 sayısı ile OR işlemine tabi tutularak imlecin 1.satırda istenen sütun değerinde olması için gereken komut değeri LCD'ye önceden tanımlanmış `lcd_komut()` fonksiyonu aracılığı ile gönderilmiştir. `if (satir==2)` komutu ile eğer satır değişkeni 2 ise `lcd_komut(0x80|(0x40+(sutun-1)));` komutunun işlem görmesi sağlanmıştır. 2x16'lık karakter tabanlı LCD'de ikinci satır başlangıç DDRAM adres değeri 0x40'tır. Bu nedenle DDRAM komutu elde edilirken sütun değerinin 1 eksiği 0x40 sayısına eklenir ve çıkan sonuç 0x80 sayısı ile OR'lanarak ilgili komut değeri bulunmuş olur. Bu sonuç yine önceden tanımlanmış `lcd_komut()` fonksiyonu ile LCD'ye gönderilir.

LCD'yi kullanmadan önce LCD ile yapılacak iletişimin kaç bit olacağı, kaçlık dot matris karakter yapısının kullanılacağı, kaç satır olduğu, imleç durumunun ne olacağı, her karakter yazımından sonra imlecin sağa mı? yoksa sola mı? kayacağı gibi bilgilerin LCD'ye bildirilmesi gerekir. Ayrıca LCD'nin ilk başta açılıp, kapatılması ve ekranın silinerek imlecin 1.satır 1. süruna getirilmesi gerekir. Bütün bu işlemler için `void lcd_hazirla()` komutu ile LCD başlangıç ayarlarının yapıldığı bir fonksiyon tanımlanmıştır. Fonksiyondan görüldüğü gibi önceden tanımlanan `lcd_komut()` fonksiyonu ile LCD'ye bu işlemler için gerekli komutlar gönderilmiştir. **lcd_hazirla() fonksiyonu diğer tüm LCD komutları kullanılmadan önce mutlaka bir kez çağrılmalıdır.**

Oluşturulan kendi LCD dosyamız ile LCD'ye karakter veya string ifadeler göndermek için C dilinde bulunan **printf ()** fonksiyonundan faydalanacaktır. Aşağıda bu işlemin

için örnek bir komut verilmiştir.

printf(lcd_veri," LCD DENEME");

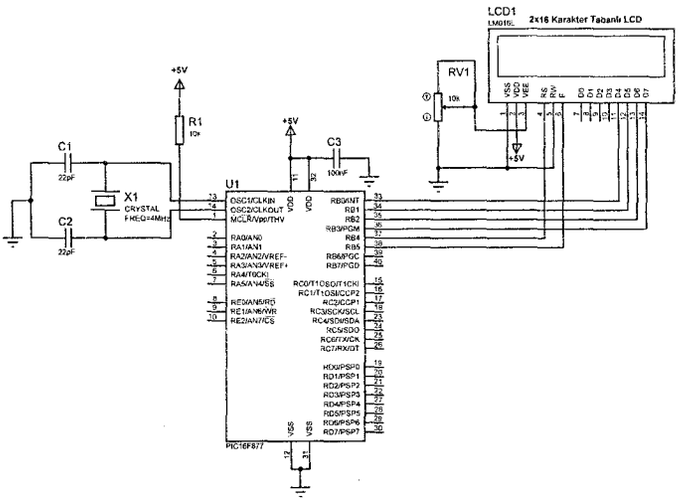
Yukarıda tanımlanan **"benim_LCD.c"** dosyasını kendi programlarınızda kullanmak için benin_LCD.c adında veya istenen bir isimde herhangi bir editör programı veya CCS C ile (File menüsünden "New" kısmına ordan "Source File" komutu ile) uzantısı ".C" olacak şekilde kaydettikten sonra **C:\Program Files\PICC\Drivers\ dizini içine kopyalamanız gerekmektedir**. Bu şekilde programınızda `#include <benim_lcd.c>` satırını programınıza yazarak dosyanızı programınıza eklemiş olursunuz. Aynı şekilde eğer dosyanızı farklı klasörde (dizinde) tutmak istiyorsanız ve bu şekilde bu dosyayı programınıza derleme işlemi için eklemek istiyorsanız şu şekilde bir komut kullanarak dosyanızın olduğu dizini programa tanıtmanız gerekmektedir. `#include <D:\Deneme\benim_LCD.c>` şeklinde.

13.7. KENDİ LCD DOSYAMIZ İLE LCD UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada kendi oluşturduğumuz "benim_lcd.c" dosyasını kullanarak LCD'ye yazı yazma uygulaması yapılmıştır. Programı denemeden önce benim_lcd.c dosyasını C:\Program Files\PICC\Drivers\ dizini içine kopyalamayı unutmayınız.

Yazdığımız dosya PIC PROG/DEKA 16F87x deneme kartı ile tümüyle uyumlu olduğundan direkt olarak deneme kartınızda deneyebilirsiniz. Tabi ki B port'u jumper'ının LCD kısmında olması gerektiğini unutmayınız.



Şekil-13.20. *Kendi LCD dosyamızla LCD uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Kendi LCD Dosyamızla LCD Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#include <benim_LCD.c> // benim_LCD.c dosyası programa ekleniyor

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   lcd_hazirla(); // LCD başlangıç ayarları yaptırılıyor

   while(1) // Sonsuz döngü
   {
      imlec(1,1); // İmlec 1.satır 1.sütunda
      printf(lcd_veri,"Kendi LCD Dosya");// LCD'ye veri yazdırılıyor
      delay_ms(1000);
      imlec(2,4); // İmlec 2.satır 4.sütunda
      printf(lcd_veri,"Denememiz"); // LCD'ye veri yazdırılıyor
      delay_ms(1000);

      lcd_komut(0x01); // LCD sıfırlanıyor
      lcd_komut(0x0E); // İmleç alt çizgi modunda
      imlec(1,6); // İmlec 1.satır 6.sütunda
      printf(lcd_veri,"C"); delay_ms(1000);
      printf(lcd_veri,"C"); delay_ms(1000);
      printf(lcd_veri,"S"); delay_ms(1000);
      printf(lcd_veri," C"); delay_ms(1000);

       lcd_komut(0x01); // LCD sıfırlanıyor
   }
}
//*****************************************************************
```

Programda dikkat edilirse önceki tüm uygulamalarda kullanılan `#use fast_io()` komutu kullanılmamıştır. Bu komut kullanıldığında programda `set_tris_x()`

komutu ile port'ların hangi bit'lerinin giriş, hangi bit'lerinin çıkış olacağının bildirilmesi gerekir. Fakat bu komut kullanılmadığında derleyici otomatik olarak `#use standart_io()` komutu kullanılmış gibi işlem yapar. Önceden bilindiği gibi bu komut geçerli olduğunda programda port pin'lerinin yönlendirilmelerinin (giriş-çıkış olma durumlarının) bildirilmesine gerek yoktur. Derleyici otomatik olarak pin yönlendirmelerini yapacaktır.

Programda `#include <benim_LCD.c>` komutu ile "benim_LCD.c" dosyası programa dahil edilmiştir. Bu komut mutlaka `#use delay (clock=4000000)` komutundan sonra kullanılmalıdır. Çünkü "benim_LCD.c" dosyasının içinde gecikme fonksiyonu kullanılmıştır. Gecikme fonksiyonu tanımlanmadan bu dosya kullanılırsa program hata verir. Ana fonksiyonda `lcd_hazirla();` komutu ile LCD başlangıç ayarlarını yapan fonksiyon çağrılmıştır. Sonsuz döngüde `imlec(1,1);` komutu ile imleç 1.satır, 1. sütuna konumlandırılmıştır. `printf(lcd_veri,"Kendi LCD Dosya");` komutu ile de LCD'ye yazı yazdırılmıştır. lcd_veri() fonksiyonu C dilinde bulunan printf() fonksiyonu ile birlikte bu şekilde kullanılmalıdır. Programda `lcd_komut(0x0E);` komutu ile imlecin alt çizgi halinde görünmesi sağlanmıştır. `lcd_komut(0x01);` komutu ile de LCD silinmiş ve imleç 1.satır 1.sütuna alınmaktadır.

13.8. LCD'DE TÜRKÇE VE ÖZEL KARAKTER KULLANMA

LCD'de sabit olan standart karakter grubu **CGROM** (Character Generator ROM) belleğinde tanımlanmışlardır. Fakat kullanıcılar en fazla 8 adet daha karakter tanımlayabilir (5x10 dot matris'e sahip LCD'lerde en fazla 4 karakter tanımlanabilir). Bu tanımlanan karakterler LCD'de **CGRAM** bellekte saklanır. Tablo-13.1'de görüldüğü gibi 0x00 ile 0x0F'e kadar olan ilk 16 karakterlik alan boştur. Bu 16 alana kullanıcılar istediği karakterleri kaydedebilir. İstenen bir karakteri kaydetmek için ilk başta CGRAM adres seçme komutu ile LCD'ye CGRAM adresi gönderilmeli ardından da karakter bilgisi gönderilmelidir. CGRAM adresi 0x40'tan 0x7F'e kadardır. Karakter bilgilerinin tespiti için ilk başta LCD'nin dot matrix yapısına (5x8 veya 5x10) göre istenen karakter oluşturulur ve bu karakterin sayısal bilgileri çıkartılır. Aşağıda 5x8 dot matrix yapıda çeşitli karakterler oluşturulmuş ve bu karakterlerin değerleri verilmiştir.

16	8	4	2	1	Değer
	■	■	■		14
■				■	17
■					16
■					16
■					16
■		■		■	21
	■	■	■		14
		■			4

16	8	4	2	1	Değer
	■	■	■		14
					0
	■	■	■	■	15
■					16
■			■	■	19
■				■	17
	■	■	■	■	15
					0

16	8	4	2	1	Değer
		■			4
					0
		■			4
		■			4
		■			4
		■			4
		■			4
					0

16	8	4	2	1	Değer
	■		■		10
					0
	■	■	■		14
■				■	17
■				■	17
■				■	17
	■	■	■		14
					0

16	8	4	2	1	Değer
	■	■	■		14
■				■	17
■					16
	■	■	■		14
				■	1
■				■	17
	■	■	■		14
		■			4

16	8	4	2	1	Değer
	■		■		10
					0
■				■	17
■				■	17
■				■	17
■				■	17
	■	■	■		14
					0

16	8	4	2	1	Değer
		■			4
		■			4
		■			4
		■			4
		■			4
		■			4
		■			4
					0

16	8	4	2	1	Değer
	■	■			12
■			■		18
■			■		18
	■	■			12
					0
					0
					0
					0

Şekil-13.21. *Oluşturulan çeşitli özel karakterler ve değerleri.*

Şekil-13.21'de görüldüğü gibi istenen karakter dot matrix'de nokta nokta oluşturulur. Daha sonra oluşturulan karakterin satır değerleri hesaplanır. Hesaplamada kolaylık olması bakımından sütun değerlerine 16, 8, 4, 2, 1 değerleri yazılmıştır. Böylece oluşturduğunuz karakterde hangi noktaların yanması istiyorsanız o sütun değerlerini toplayıp satır değerini elde edebilirsiniz. Görüldüğü gibi 5x8 bir dot matrix LCD'de bir karakter için 8 adet bilgi gereklidir. İstenen tüm özel karakterler bu mantık çerçevesinde oluşturulup, veri değerleri hesaplanabilir.

CGRAM'a oluşturulan özel karakterleri kaydetmek için ilk başta CGRAM adres belirleme komutu ile CGRAM adresi belirlenir. Tablo-13.2'ye bakılırsa bu komutun 1 AC5 AC4 AC3 AC2 AC1 AC0 formatında olduğu görülür. En baştaki 1 bilgisi hep sabittir. Bu komuta göre CGRAM özel karakter adresi en düşük 0x40 en fazla 0x7F'dir. Her tek bir CGRAM adresine 8 adet komut ve karakter satır verisi yollanır. Bu durumda CGRAM adreslerine yerleştirilen karakterlerin satır bilgiler için komut değerleri şunlar olur.

CGRAM 0. Adresi → 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47
CGRAM 1. Adresi → 0x48, 0x49, 0x4A, 0x4B, 0x4B, 0x4D, 0x4E, 0x4F
CGRAM 2. Adresi → 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57
CGRAM 3. Adresi → 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F
CGRAM 4. Adresi → 0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67
.....

LCD'ye ilk başta CGRAM adres komutu gönderilir ardından da karakterin ilk satır değeri gönderilir. Bu işlem bu şekilde devam eder.

NOT: CGRAM'a kaydedilen özel karakterler enerji kesilince LCD hafızasından silinir. Bu nedenle LCD her kullanıldığında özel karakter bilgilerinin başlangıçta LCD'ye yüklenmesi gerekir.

13.9. ÖZEL KARAKTERLİ LCD DOSYASI OLUŞTURMA

Daha önce oluşturduğumuz LCD dosyasında Türkçe karakterleri kullanamıyorduk. Çünkü Türkçe karakterler standart LCD karakter grubunda bulunmamaktadır. Ayrıca çeşitli uygulamalarda çeşitli özel karakterleri LCD'de göstermek isteyebilirsiniz. Bu nedenle önceden oluşturduğumuz benim_LCD.c dosyasına bazı eklemeler yaparak Türkçe karakterleri ve özel karakterleri tanımlayabilen **"özel_karakter_LCD.c"** dosyasını oluşturacağız. **Bu dosyada tanımlanan karakterleri değiştirebilir veya kendiniz de başka özel karakterler ekleyebilirsiniz.**

özel_karakter_LCD.c Dosyasının İçeriği:

```
//*****************************************************************
//////////////////////////////////////////////////////////////////
//                    özel_karakter_LCD.c                    //////
// Bu LCD sürücü dosyası 2x16 HD44780 uyumlu LCD'ler          /////
// için yazılmıştır. LCD'den sadece okuma yapılacağı için     /////
// R/W ucu şaseye bağlanmalıdır. LCD bağlantısı aşağıdadır. /////
// Bu LCD dosyası ile özel karakterler de LCD gösterilebilir ////
//////////////////////////////////////////////////////////////////
// RB0 D4
// RB1 D5
// RB2 D6
// RB3 D7
// RB4 RS
// RB5 E
// R/W ucu direkt şaseye bağlanacak

#define e   pin_b5 // LCD'nin E ucu RB5 pin'ine bağlı
#define rs  pin_b4 // LCD'nin RS ucu RB4 pin'ine bağlı

//****** LCD'ye Komut Gönderme Fonksiyonu **********
void lcd_komut(byte komut)
{
   output_b(komut>>4);  // Komutun yüksek değerli 4 bit'ini gönder
   output_low(rs);      // LCD komut almak için ayarlandı
   output_high(e);      // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(2);         // 2 msn gecikme veriliyor

   output_b(komut&0x0F); // Komutun düşük değerli 4 bit'ini gönder
   output_low(rs);       // LCD komut almak için  yarlandı
   output_high(e);       // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(2);          // 2 msn gecikme veriliyor
}
```

```
//******* LCD'ye Veri Gönderme Fonksiyonu **********
void lcd_veri(byte veri)
{
if(veri=='Ç') veri=0;//Karakter Ç ise CGRAM 0.adresteki bilgiyi gönder
if(veri=='Ğ') veri=1;//Karakter Ğ ise CGRAM 1.adresteki bilgiyi gönder
if(veri=='İ') veri=2;// Karakter İ ise CGRAM 2.adresteki bilgiyi gönder
if(veri=='Ö') veri=3;// Karakter Ö ise CGRAM 3.adresteki bilgiyi gönder
if(veri=='Ş') veri=4;// Karakter Ş ise CGRAM 4.adresteki bilgiyi gönder
if(veri=='Ü') veri=5;// Karakter Ü ise CGRAM 5.adresteki bilgiyi gönder
if(veri=='I') veri=6;// Karakter I ise CGRAM 6.adresteki bilgiyi gönder
if(veri==''') veri=7;// Karakter ⁰ ise CGRAM 7.adresteki bilgiyi gönder

   output_b(veri>>4);    // Verinin yüksek değerli 4 bit'ini gönder
   output_high(rs);      // LCD veri almak için ayarlandı
   output_high(e);       // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(1);          // 1 msn gecikme veriliyor

   output_b(veri&0x0F); // Verinin düşük değerli 4 bit'ini gönder
   output_high(rs);      // LCD veri almak için ayarlandı
   output_high(e);       // E ucu lojik-1'den lojik-0'a çekiliyor
   output_low(e);
   delay_ms(1);          // 1 msn gecikme veriliyor
}

//******* LCD'de İmlec Konumlandırma Fonksiyonu ********
void imlec(byte satir, byte sutun)
{
   if (satir==1)          // Eğer satır değişkeni "1" ise
      lcd_komut(0x80|(sutun-1));

   if (satir==2)          // Eğer satır değişkeni "2" ise
      lcd_komut(0x80|(0x40+(sutun-1)));
}

//********* LCD Başlangıç Ayarları Fonksiyonu ******
void lcd_hazirla()
{
   lcd_komut(0x02);// LCD'yi kullanıma hazır hale getir, imleç
                   // 1.satır 1.sütunda komutu
   lcd_komut(0x28);//2 satır, 4 bit iletişim, 5x8 dot matris seçildi
   lcd_komut(0x08);//Display kapalı,alt çizgi ve yanıp sönme yok
   lcd_komut(0x0C);//Display açık,imleç alt çizgi ve yanıp sönme yok
   lcd_komut(0x06);// Her veri yazıldığında imleç bir sağa gitsin
   lcd_komut(0x01);// Display sil. İmleç 1.satır 1.sütunda
   delay_ms(1);    // 1 msn bekle
}

//*** Özel Karakterleri CGRAM'e Yükleyen Fonksiyon ****
void ozel_karakterler()
{
   // CGRAM'de 1. Adrese "Ç" Karakteri Kaydediliyor
```

```
lcd_komut(0x40);lcd_veri(14);
lcd_komut(0x41);lcd_veri(17);
lcd_komut(0x42);lcd_veri(16);
lcd_komut(0x43);lcd_veri(16);
lcd_komut(0x44);lcd_veri(16);
lcd_komut(0x45);lcd_veri(21);
lcd_komut(0x46);lcd_veri(14);
lcd_komut(0x47);lcd_veri(4);

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

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

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

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

// CGRAM'de 5. Adrese "Ü" Karakteri Kaydediliyor
lcd_komut(0x68);lcd_veri(10);
lcd_komut(0x69);lcd_veri(0);
```

```
   lcd_komut(0x6A);lcd_veri(17);
   lcd_komut(0x6B);lcd_veri(17);
   lcd_komut(0x6C);lcd_veri(17);
   lcd_komut(0x6D);lcd_veri(17);
   lcd_komut(0x6E);lcd_veri(14);
   lcd_komut(0x6F);lcd_veri(0);

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

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

Dosyada veri gönderme fonksiyonuna önceki oluşturduğumuz dosyaya göre ek komutlar eklenmiştir. LCD'ye veri gönderme fonksiyonunda `if(veri=='Ç') veri=0;` komutu ile LCD'ye yazdırılacak veride eğer "Ç" harfi varsa o zaman CGRAM'de 0. adrese yazılan Ç harfi bilgisinin LCD'ye gönderilmesi sağlanmıştır. Diğer if komutları ile de aynı mantık ile işlem yapılmıştır.

Bu dosyada ki en önemli farklılık ise oluşturulan özel karakterleri CGRAM'e yükleyen fonksiyondur. Bu fonksiyon `void ozel_karakterler()` komutu ile tanımlanmıştır. Fonksiyonda CGRAM'in 0.,1.,2.,3.,4.,5.,6. ve 7. adreslerine çeşitli özel karakterler kaydedilmiştir. Örneğin `lcd_komut(0x40);` komutu ile CGRAM 0.adresine kaydedilecek karakterin ilk satır adres bilgisi LCD'ye gönderilmiştir. Ardından da `lcd_veri(14);` komutu ile de bu adrese ilgili karakterin ilk satır bilgisi gönderilmiştir. Bu işlem sırayla karakterin son satır değerini gönderene kadar aynı şekilde uygulanmıştır. Diğer tüm özel karakter bilgileri de bu şekilde CGRAM'e yüklenmiştir.

Programınızda LCD komutlarını kullanmadan önce `lcd_hazirla()` fonksiyonunu çağırmanız ve ardından da özel karakterlerin CGRAM'e yüklenmesi için `ozel_karakterler()` fonksiyonunu çağırmalısınız.

13.10. ÖZEL KARAKTERLİ LCD UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada oluşturulan "özel_karakter_LCD.c" dosyası programa eklenerek LCD'ye yazı yazdırma uygulaması incelenmiştir. Devre şeması olarak bir önceki uygulamadaki Şekil-13.20'deki devre şemasının aynısı kullanılmıştır.

↳ Oluşturduğumuz "özel_karakter_LCD.c" dosyasını C:\Program Files\PICC\Drivers\ dizini içine kopyalamayı unutmayınız.

Devre programı;

```
/*****************************************************************
PIC16F877 ile Kendi Özel Karakterli LCD Dosyamızla LCD Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#include <özel_karakter_LCD.c> // özelkarakter__LCD.c dosyası
                               // programa ekleniyor
float x=23.5;             // Ondalıklı tipte değişken tanımlanıyor
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   lcd_hazirla();       // LCD başlangıç ayarları yaptırılıyor
   ozel_karakterler();  // Özel karakterler CGRAM'e yükleniyor

   while(1) // Sonsuz döngü
   {
      imlec(1,1); // İmlec 1.satır 1.sütunda
      printf(lcd_veri,"ÖZEL KARAKTERLİ");// LCD'ye veri yazdırılıyor

      imlec(2,1); // İmlec 2.satır 4.sütunda
      printf(lcd_veri,"LCD UYGULAMASI"); // LCD'ye veri yazdırılıyor
      delay_ms(1000);

      lcd_komut(0x01); // LCD sıfırlanıyor
      printf(lcd_veri,"ÖZEL KARAKTERLER");//LCD'ye veri yazdırılıyor
```

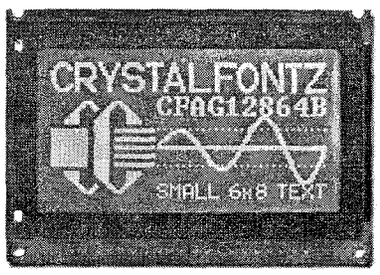
```
        imlec(2,1); // İmlec 2.satır 1.sütunda
        printf(lcd_veri,"Ç"); delay_ms(1000);
        printf(lcd_veri,"Ğ"); delay_ms(1000);
        printf(lcd_veri,"İ"); delay_ms(1000);
        printf(lcd_veri,"Ö"); delay_ms(1000);
        printf(lcd_veri,"Ş"); delay_ms(1000);
        printf(lcd_veri,"Ü"); delay_ms(1000);
        printf(lcd_veri,"'"); delay_ms(1000);
        printf(lcd_veri,"I"); delay_ms(1000);

        lcd_komut(0x01); // LCD sıfırlanıyor
        printf(lcd_veri,"SICAKLIK %f'C",x);
         delay_ms(3000);
        lcd_komut(0x01); // LCD sıfırlanıyor
    }
}
/*****************************************************************
```

"Özel_karakter_LCD.c" dosyası `#include <özel_karakter_LCD.c>` komutu ile programa eklenmiştir. Programda dikkat edilirse hiçbir LCD komutu kullanılmadan ilk başta LCD başlangıç ayarları için `lcd_hazirla();` komutu kullanılarak LCD başlangıç ayarlarını yapan fonksiyon çağrılmıştır. Ardından da özel karakterlerin CGRAM'e yüklenmesi için `ozel_karakterler();` komutu ile bu fonksiyon çağrılmıştır.

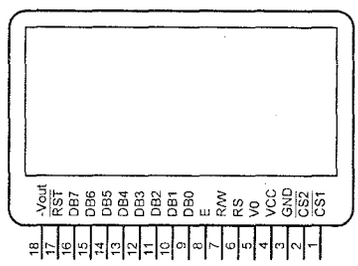
Programda en son kullanılan `printf(lcd_veri,"SICAKLIK %f'C",x);` komutuna dikkat edilirse LCD'ye "SICAKLIK" yazısı yazdırılmış. Ardından "%f" komutu ile buraya ondalıklı tipte bir değişken geleceği belirtilmiştir. Komutta en son çift tırnak işaretinden sonra virgül(,) konulmuş ve bahsedilen değişken ismi yazılmıştır.

13.11. GRAFİK LCD

Grafik LCD'ler karakter tabanlı LCD'lerin tüm özelliklerini yerine getirebilirler. Bunun yanında grafik LCD'lerde piksel düzeyinde kontrol imkânı olduğundan, grafik LCD'ler istenen çizimlerde gösterilebilir. Grafik LCD'lerde karakter tabanlı LCD'lerde olduğu gibi kontrol entegreleri içerirler. Grafik LCD'lerin kontrol edilmesi işlemi daha zordur.



Şekil.13.22. Grafik LCD.



Şekil-13.23. Proteus Isis'de Grafik LCD.

Grafik LCD'lerin farklı kontrol entegreleri mevcuttur. Piyasada K0108 entegreleri ile uyumlu birçok grafik LCD rahatlıkla bulunabilir. Tablo-13.8de K0108 entegreli HDM64GS12 grafik LCD'nin pin isimleri ve açıklamaları verilmiştir. Verilen pin numaraları çeşitli grafik LCD modellerine göre değişiklik gösterebilir. Bu nedenle doğru pin bağlantısı için mutlaka kullandığınız grafik LCD'nin teknik dokümanına bakmanız gereklidir.

Pin No	Pin İsmi	Açıklama
1	VSS (GND)	Şase
2	VDD (Vcc)	+5V
3	Vo	Parlaklık Ayarı
4	D/I	Data veya Komut Giriş Seçme Ucu
5	R/W	Data Okuma/Yazma
6	E	Yetki verme ucu
7	DB0	Veri Uçlarıdır. Bu uçlardan hem veri hem de komut kodu gönderilir.
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	CS1	IC1 seçme ucu
16	CS2	IC2 seçme ucu
17	RES (RST)	Reset ucu
18	VEE (-Vout)	Negatif Voltaj Çıkışı
19	K	Arka ışık GND ucu
20	A	Arka ışık +5V ucu

Tablo-13.8. *KS0108 entegreli HDM64GS12 grafik LCD pin açıklamaları.*

13.11.1. CCS C Grafik LCD Kütüphane Dosyaları

CCS C programı içinde grafik LCD ile ilgili 4 adet kütüphane dosyası bulunmaktadır. Bu dosyalar C:\Program Files\PICC\Drivers\ klasörü içinde bulunmaktadır. Grafik LCD ile ilgili CCS C dosyaları şunlardır;

- GLCD.c
- HDM64GS12.c
- GRAPHICS.c
- KS0108.c

GLCD.c dosyası, Hantronix HDM64GS12 grafik LCD ile ilgili kontrol fonksiyonlarını içerir. 128x64 Grafik LCD'ler için uyumludur.
HDM64GS12.c dosyası, KS0108 LCD kontrolcüsüne sahip Hantronix HDM64GS12 grafik LCD ile ilgili kontrol fonksiyonları içerir. 128x64 Grafik LCD'ler için uyumludur.
GRAPHICS.c dosyası, grafik LCD'lerde çizgi, dikdörtgen, daire, bar çubuk ve karakter gösterimleri ile ilgili fonksiyonları içerir.

KS0108.c dosyası, Ks0108 LCD kontrolcüsü ile ilgili kontrol fonksiyonlarını içerir.

Bahsedilen dosyaların içinde geçen fonksiyonların bir çoğu aynı isimdedir ve aynı işlemleri yaparlar. İstenirse CCS C için yazılmış başka grafik LCD fonksiyon dosyaları CCS C forum sitesinden ve diğer internet kaynaklarından da bulunabilir. Aşağıda GLCD.c, HDM64GS12.c ve GRAPHICS.c dosyalarının içerdiği fonksiyonlar ve kullanımları verilmiştir. Dosyalar içinde bulunan aynı isimli fonksiyonlar aynı görevi yerine getirirler.

GLCD.c ve HDM64GS12.c dosyalarında grafik LCD bağlantısı aşağıda verilmiştir.

Grafik LCD bağlantı pin'leri şunlardır. D port'una bağlı olan data hatları hariç, diğer uçlar bu dosya içinde değişiklik yapılarak farklı pin'ler kullanılabilir.

RB0	CS1	**RD0**	DB0
RB1	CS2	**RD1**	DB1
RB2	DI	**RD2**	DB2
RB4	R/W	**RD3**	DB3
RB5	E	**RD4**	DB4
RC0	RST	**RD5**	DB5
		RD6	DB6
		RD7	RD7

GLCD.c Dosyası Fonksiyonları

glcd_init(mode) = Diğer tüm grafik LCD komutları kullanılmadan önce tanımlanması gerekir. Fonksiyondaki *"mode"* kısmına *"ON"* veya *"OFF"* sabitleri yazılabilir. "ON" LCD'yi açar, OFF kapatır.
glcd_pixel(x,y,color) = Belirtilen x ve y koordinatlarındaki pikseli aktif veya pasif etmeye yarar. "x" ve "y" değişkenleri koordinatları belirtir. *"color"* değişkeni ise *ON* veya *OFF* değerlerini alarak belirtilen yerdeki pikselin siyah veya beyaz olmasını belirler. ON siyah, OFF beyaz görevini görür.
glcd_line(x1, y1, x2, y2, color) = Verilen iki nokta arasında çizgi çizer. Fonksiyondaki *x1, y1* değişkenleri ilk noktanın konumunu *x2,y2* değişkenleri ikinci noktanın konumunu belirtir. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değişkeni ile çizgi siyah, "OFF" değişkeni ile çizgi beyaz olur.
glcd_rect(x1, y1, x2, y2, fill, color) = Belirlenen 2 köşe noktası arasında dikdörtgen çizme işlemini yapar. Fonksiyondaki *x1, y1* değişkenleri ilk köşenin konumunu *x2,y2* değişkenleri ikinci köşenin konumunu belirtir. *"fill"* değişkeni *"YES"* ve *"NO"* sabitlerini alabilir. "YES" sabiti ile çizilen dikdötgenin içi boyanır, "NO" sabiti ile çizilen dikdörtgenin içi boyanmaz. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değişkeni ile dikdörtgen siyah, "OFF" değişkeni ile dikdörtgen beyaz olur.
glcd_bar(x1, y1, x2, y2, width, color) = Belirlenen iki nokta arasında çubuk (bar) çizer. Fonksiyondaki *x1, y1* değişkenleri ilk noktanın konumunu *x2,y2*

değişkenleri ikinci noktanın konumunu belirtir. Fonksiyondaki *"width"* değişkeni çizdirilen çubuğun piksel olarak genişliğini belirtir. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değişkeni ile çubuk siyah, "OFF" değişkeni ile beyaz olur.

glcd_circle(x, y, radius, fill, color) = Belirlenen merkez noktası etrafında belirlenen yarıçapta daire çizer. Fonksiyondaki *x* ve *y* değişkenleri daire merkezini, *"radius"* değişkeni dairenin yarıçapını belirtir. *"fill"* değişkeni *"YES"* ve *"NO"* sabitlerini alabilir. "YES" sabiti ile çizilen dairenin içi boyanır, "NO" sabiti ile çizilen dairenin içi boyanmaz. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değişkeni ile daire siyah, "OFF" değişkeni ile daire beyaz olur.

glcd_text57(x, y, textptr, size, color) = Bu fonksiyon ile *x* ve *y* koordinatları ile belirlenen noktadan başlanarak, *"textptr"* string dizisindeki karakterleri, *"size"* değişkeni ile belirtilen piksel büyüklüğünde display'de yazmaya yarar. Ekrana yazdırılan karakterler 5 piksel genişliğinde, 7 piksel yüksekliğindedir. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değişkeni ile daire siyah, "OFF" değişkeni ile daire beyaz olur. Bu fonksiyon ile ekrana yazdırılan yazılar ekran sınırını aşarsa otomatik olarak bir alt satıra geçilir.

glcd_fillScreen(color) = Tüm LCD ekranını karartır veya karartılmış ekranı normal haline geri döndürür. *"color"* değişkeni *"ON"* veya *"OFF"* değerlerini alabilir. "ON" değeri ile ekran tümüyle karartılır. "OFF" değeri ile karartılmış ekran eski haline getirilir.

HDM64GS12.c Dosyası Fonksiyonları

#define FAST_GLCD = Eğer kullanılan PIC denetleyici RAM hafızası fazla olan denetleyiciler de kullanılabilir (18F serisi için uygundur). Bu komutun program başında HDM64GS12.c dosyası tanıtılmadan önce kullanılması gerekir. Bu komut ile hızlı bir LCD display'inin tazelenmesi işlemi için RAM hafızası yoğunlukla kullanılır.

glcd_update() = RAM bellekte saklanmış display bilgilerini LCD'ye yazar. Bu komut sadece #define fast_GLCD komutu kullanılmış ise kullanılır.

glcd_init(mode), glcd_pixel(x,y,color), glcd_fillScreen(color) fonksiyonları da bu dosya içindedir. Bu fonksiyonların görevleri GLCD.c dosyası tanıtılırken belirtilmişti.

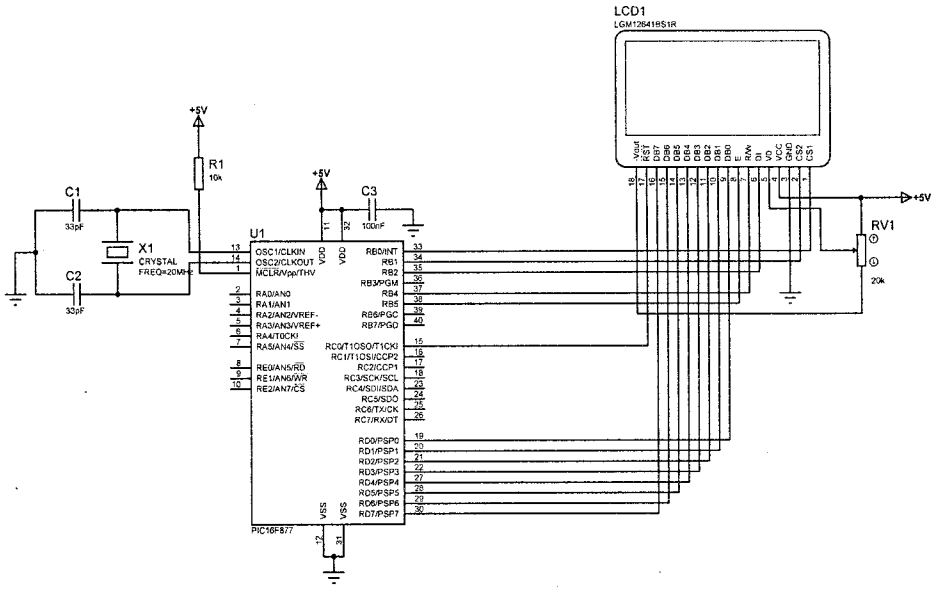
GRAPHICS.c Dosyası Fonksiyonları

GRAPHICS.c dosyası şu fonksiyonları içerir. **glcd_line(x1, y1, x2, y2, color), glcd_rect(x1, y1, x2, y2, fill, color), glcd_bar(x1, y1, x2, y2, width, color), glcd_circle(x, y, radius, fill, color), glcd_text57(x, y, textptr, size, color).** Bu fonksiyonlar daha önce GLCD.c dosyası tanıtılırken belirtilmişti. GRAPHICS.c dosyası geometrik şekillerin çizilmesi ile ilgili ayrı bir dosyadır. Bu dosya kullanılan bir grafik LCD kontrol dosyası ile birlikte kullanılabilir. Böylece geometrik şekiller için ayrıca fonksiyon yazmaya gerek kalmaz.

13.12. GRAFİK LCD UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada KS0108 kontrol entegreli HDM64GS12 grafik LCD üzerinde yazı yazdırma, çizgi çizme, çubuk çizme ve daireler oluşturma işlemleri ile yukarıda anlatılan dosyalardaki fonksiyonların kullanımına örnek oluşturulmuştur.



Şekil-13.24. *Grafik LCD uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Grafik LCD Uygulaması
*****************************************************************/

#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses HS,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları
#use delay (clock=20000000) // Gecikme fonksiyonu için kullanılacak
                            // osilatör frekansı belirtiliyor.
#include <HDM64GS12.c> // HDM64GS12.c dosyası programa ekleniyor
#include <graphics.c>  // graphics.c dosyası programa ekleniyor

char yazi1[]="GAZI";          // Karakter dizisi tanımlanıyor
char yazi2[]="UNIVERSITY";    // Karakter dizisi tanımlanıyor
int i,x1,y1,x2,y2;
```

```
/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   glcd_init(ON); // Grafik LCD hazırlanıyor ve ekran siliniyor

   while(1)
   {
      glcd_init(ON); // Ekran siliniyor
      glcd_text57(39, 5, yazi1, 2, ON); // GLCD'de yazı yazdırılıyor
      glcd_text57(5, 30, yazi2, 2, ON); // GLCD'de yazı yazdırılıyor
      delay_ms(2000);

      glcd_init(ON); // Ekran siliniyor
      x1=5;
      y1=5;
      x2=100;
      y2=5;
      for(i=0;i<6;i++)
      {
         glcd_line(x1, y1, x2, y2, ON); //GLCD'de Çizgi çizdiriliyor
         y2=y1+=10; // y2=y1=y2+10; anlamında
         x2-=10;    // x2=x2-10; anlamında
         delay_ms(1000);
      }
      glcd_init(ON); // Ekran siliniyor
      x1=y1=10;
      for (i=0;i<120;i++)
      {
         glcd_pixel(x1,y1,ON); // Ekranda istenen pikseller aktif
                               // yapılıyor
         y1=x1++;   // y1=x1+1; anlamında
         delay_ms(50);
      }

      glcd_init(ON); // Ekran siliniyor
      y2=60;
      x1=5;
      for (i=0;i<8;i++)
      {
         glcd_bar(x1,0,x1,y2,10,ON); // GLCD'de çubuk çizdiriliyor
         delay_ms(1000);
         x1+=15; // x1=x1+15; anlamında
         y2-=5;  // y2=y2-5; anlamında
```

```
        }

        glcd_init(ON); // Ekran siliniyor
        for(i=0;i<=30;i=i+5)
        {
           glcd_circle(60,30,i,NO,YES); // GLCD'de Daire çizdiriliyor
           delay_ms(1000);
        }

        glcd_init(ON); // Ekran siliniyor
        x1=5;
        y1=5;
        x2=120;
        y2=63;
        for(i=0;i<6;i++)
        {
           glcd_rect(x1,y1,x2,y2,NO,YES); // GLCD'de dikdörtgen
                                          // çizdiriliyor
           y1=x1+=5; // y1=x1+5; anlamında
           x2-=5;    // x2=x2-5; anlamında
           y2-=5;    // y2=y2-5; anlamında
           delay_ms(500);
        }
    }
}
//*****************************************************************
```

Programda ilk başta kullanılan grafik LCD için HDM64GS12.c dosyası `#include <HDM64GS12.c>` komutu ile programa dâhil edilmiştir. Grafik LCD'de çizgi ve şekiller çizmek içinde GRAPHICS.c dosyası `#include <graphics.c>` komutu ile programa dahil edilmiştir. Grafik LCD'de karakter oluşturmak için mutlaka yazdırılacak karakterleri içinde barındıran bir dizi tanımlanmalıdır. Bu nedenle `char yazi1[]="GAZI";` ve `char yazi2[]="UNIVERSITY";` komutlarıyla karakter dizileri tanımlanmıştır. Ana fonksiyondan önce `glcd_init(ON);` fonksiyonu ile grafik LCD başlangıç ayarlarının yapılması sağlanmış ve ekran temizlenmiştir. Bu fonksiyon aynı zamanda ekran temizlenmesi amacıyla da kullanılır.

Sonsuz döngüde ekran temizlendikten sonra `glcd_text57(39, 5, yazi1, 2, ON);` fonksiyonu ile LCD'ye yazi1[] dizisi yazdırılmıştır. Bu dizi ekrana (39,5) konumundan başlanarak yazılmaya başlanacaktır. Ayrıca fonksiyondaki "2" değişkeni ile karakter büyüklüğü belirtilmiştir. Eğer ekrana yazdırılacak karakterler çok büyük olursa ve LCD'ye sığmasa sığmayan karakterler otomatik olarak bir alta yazdırılır. Fonksiyondaki "ON" değişkeni ile de karakterlerin rengi siyah yapılmıştır.

`glcd_line(x1, y1, x2, y2, ON);` fonksiyonu ile LCD'ye çizgi çizdirilmiştir. Ekrana `for(i=0;i<6;i++)` döngüsünü kullanarak LCD'de 6 adet çizgi çizdirilmiştir. Döngü öncesinde x1, y1, x2, y2 değişkenlerine değer verilmiştir. Döngü içinde bu değişkenlerin değerleri değiştirilerek ekrana alt alta sırayla küçülen boyda çizgi çizdirilmiştir. Fonksiyondaki "ON" değişkeni ile de karakterlerin rengi

siyah yapılmıştır. Bu işlemler yapılmadan önce `glcd_init(ON);` fonksiyonu ile ekranın temizlendiğine dikkat edin.

`glcd_pixel(x1,y1,ON);` fonksiyonu ile ekranda istenen pikseller aktif edilmiştir. Yapılan döngü ile piksel konumları aşağı doğru paralel çizgi oluşturacak şekilde değiştirilmektedir. "ON" değişkeni ile de karakterlerin rengi siyah yapılmıştır.

`glcd_bar(x1,0,x1,y2,10,ON);` fonksiyonu ile de ekrana çubuk çizdirilmiştir. Bir önceki işlemde olduğu gibi burada da `for(i=0;i<=30;i=i+5)` fonksiyonu ile oluşturulan döngü ile ekrana 8 adet azalan boyda çubuk çizdirilmiştir. Fonksiyondaki "10" sayısı çubukların genişliğini vermektedir.

`glcd_circle(60,30,i,NO,YES);` fonksiyonu ile de önceki çizimlerde kullanılan aynı mantık ile ekrana içten dışa büyüyen daireler çizdirilmiştir. Fonksiyonda ki i değişkeni dairenin yarıçapını ifade etmektedir. "NO" sabiti ise dairelerin içinin doldurulmaması içindir. Eğer içi dolu daire çizdirilmek istenseydi "OFF" sabiti kullanılırdı.

`glcd_rect(x1,y1,x2,y2,NO,YES);` fonksiyonuyla da içi boş dikdörtgen çizdirilmiştir. Fonksiyonda "NO" sabiti ile çizdirilen dikdörtgenlerin içinin boş olması sağlanmıştır. "YES" sabiti ile de çizilen dikdörtgenlerin siyah olması belirtilmiştir. Kullanılan for döngüsü sayesinde ekranda dıştan içe doğru küçülen dikdörtgenler oluşturulmuştur.

Her yeni çiziminde ekranın temizlenmesi için `glcd_init(ON);` komutunun kullanıldığına dikkat edin. Eğer bu komut kullanılmasaydı gösterilen bütün çizimler, karakterler ekranda sabit kalacak ve iç içe geçerek görünümler karışacaktı.

14. Bölüm

ANALOG/DİJİTAL ÇEVİRİCİ MODÜLÜ İŞLEMLERİ

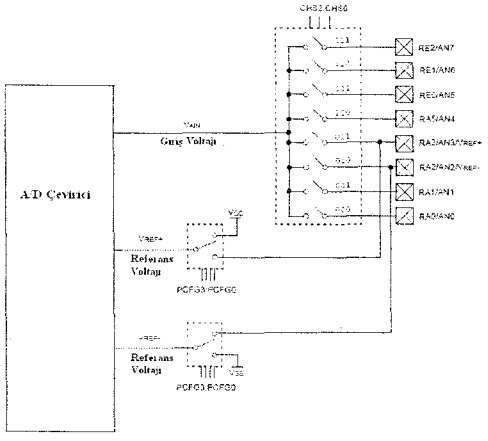
- A/D Modülü
- A/D Kesmesi
- SETUP_ADC() Fonksiyonu
- SETUP_ADC_PORTS () Fonksiyonu
- SET_ADC_CHANNEL() Fonksiyonu
- READ_ADC () Fonksiyonu
- #DEVICE ADC=X Komutu
- ADC Uygulaması
- LM35 Sıcaklık Sensörü
- LM35 Uygulaması

Bu bölümde PIC16F877 denetleyicisi içinde bulunan ADC modülü (Analog Digital Convertor - Analog Dijital Çevirici) incelenmiştir. Diğer PIC ürünlerinde de bulunan ADC modülü benzer özelliktedirler. Ardından CCS C programında ADC ile ilgili fonksiyonlar tanıtılmış ve bir ADC uygulaması gerçekleştirilmiştir. Bu bölümde aynı zamanda sıcaklık değerinin analog sinyal şeklinde veren LM35 ısı sensörü tanıtılmış ve bu sensör ile ilgili bir uygulama gerçekleştirilmiştir.

14.1. A/D MODÜLÜ

Doğada doğal olarak bulunan tüm sistemler analog düzendedir. Kullandığımız mikrodenetleyiciler ise dijital sistemlerdir. Bu nedenle dış dünya ile iletişim kurmak ve dış dünyadan veri almak -algılama yapmak- için bu iki sistemin bir şekilde birbirini anlaması gereklidir. Bu iş içinde doğada bulunan analog sinyallerin (ısı, ışık, ses, nem vb.) dijital sistemlerin anlayacağı dijital sinyallere çevrilmesi gereklidir. Bu işlem için ADC devreleri ve entegreleri kullanılmaktadır. PIC16F877

mikrodenetleyicisinde ADC modülü bulunmaktadır. Böylece harici bir devre veya entegre kullanmadan analog sinyaller dahili ADC modülü sayesinde dijital sinyallere çevrilebilir. Şekil-14.1'de PIC16F877'de bulunan ADC modülünün blok diyagramı görülmektedir.



Şekil-14.1. *A/D Modülü blok diyagramı.*

PIC16F877 8 ADC giriş ucu (RA0/AN0, RA1/AN1, RA2/AN2, RA3/AN3, RA5/AN4, RE0/AN5, RE1/AN6, RE2/AN7) vardır. ADC modülünün çözünürlüğü 10 bit'tir. ADC birimi için gerekli referans voltajı V_{DD}, V_{SS}, RA2 ve RA3 uçları ve/veya bunların kombinasyonlarından seçilerek elde edilir. **ADC birimi 4 adet kaydediciye sahiptir. ADC çevrim sonucunun yazıldığı ADRESH (ADRES High Register), ADRESL (ADRES Low Register). ADC birimi kontrol kaydedicileri ise ADCON0 (A/D Control Register0) ve ADCON1'dir (A/D Control Register1)** . ADCON0 kaydedicisi ve bit'lerinin açıklamaları şekil-14.2'de, ADCON1 kaydedicisi ve bit'lerinin açıklamaları ise şekil-14.3'de görülmektedir.

ADCON0 KAYDEDİCİSİ

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/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

Şekil-14.2. *ADCON0 kaydedicisi.*

Bit 7-6 **ADCS1:ADCS0** = A/D Çevrim Clock Seçme Bit'i.
00 = $F_{OSC}/2$
01 = $F_{OSC}/8$
10 = $F_{OSC}/32$
11 = F_{RC} (Dahili RC Osilatör)

Bit 5-3 **CHS2:CHS1** = Analog Sinyal Giriş Kanalı Seçme Bit'i. Seçilen uçtaki sinyalin analog değeri dijital bilgiye çevrilir.
000: Kanal 0, (RA0, AN0)

001: Kanal 1, (RA1, AN1)
010: Kanal 2, (RA2, AN2)
011: Kanal 3, (RA3, AN3)
100: Kanal 4, (RA4, AN4)
101: Kanal 5, (RA5, AN5)
110: Kanal 6, (RA6, AN6)
111: Kanal 7, (RA7, AN7)

Bit 2 **GO/DONE** = A/D Çevrim Durum Bit'i
Eğer ADON bit'i 1 ise;
1: A/D modülü işlem yapıyor. (ADC çevrimine başlamak için ayarlanır.)
0: A/D modülü işlem yapmıyor.(Bu bit ADC çevrimi bitince otomatik olarak 0 olur.)

Bit 1 Kullanılmayan bit. Okunduğunda "0" değeri alınır.

Bit 0 **ADON** = A/D On Bit'i
1: ADC modülü açık
0: ADC modülü kapalı

R= Okunabilir bit, **W=** İlgili bit'e bilgi yazılabilir, **U=** Kullanılmayan bit, **-0=** İlk enerji geldiğinde "0" olarak okunur, anlamındadır.

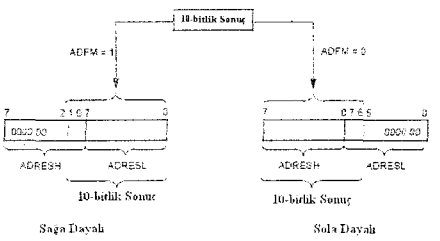
ADCON1 KAYDEDİCİSİ

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

bit 7 bit 0

Şekil-14.3. *ADCON1 kaydedicisi.*

Bit 7 **ADFM** = A/D çevirimi sonuç Formatı Yazılma Seçenekleri
1: ADRESH ve ADRESL kaydedicileri 8'er bit'tir. Toplam 16 bit'lik sayıyı muhafaza edebilirler. ADC çevrim sonucu 10 bit'lik sayı olarak bu kaydedicilere yazılır. Bu bit lojik-1 olunca sonuç **sağa dayalı** olarak yazılır. Yani ADRESH kaydedicisi içindeki 6 yüksek değerlikli bit'in değeri 0'dır. Çünkü sonuç sağa dayalı yazılmıştır.
0: ADC çevrim sonucu sola dayalı olarak yazılır. Yani ADRESL kaydedicisi içindeki 6 az değerlikli bit'in değeri 0'dır.



Bit 6-4 = Kullanılmayan bit'ler. Okunduğunda "0" değeri alınır.

Bit 3-0 **PCFG3:PCFG0** = ADC Analog Uç Ayar Bit'leri. Aşağıdaki tabloda bu bit'lere verilen değerlere göre analog uçların hangilerinin voltaj referans gerilim ucu, hangilerinin analog sinyal giriş ucu olacağı belirlenir.

PCFG3: PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	Kanal/ REF
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	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	RA3	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	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog giriş D = Dijital I/O

10 bit'lik bir ADC en fazla 2^{10} = 1024 adet değer ile bir analog işareti örnekleyebilir. ADC biriminin elde ettiği dijital bilginin bit sayısı ADC modülünün **çözünürlüğünü** ifade eder. Çözünürlük ne kadar yüksekse o kadar iyi bir dönüşüm yapılır. Örneğin; ADC'nin dijital bilgiye çevireceği sinyalin maksimum gerilim değeri yani belirlenen V_{REF} değeri 5V ise ADC **adım büyüklüğü** şu şekilde hesaplanır.

$$\text{Adım Büyüklüğü} \quad \frac{\text{Sinyalin Maksimum Gerilim Değeri}}{2^{\text{ADC Bit Sayısı}}} \quad \frac{5V}{2^{10}} \quad \frac{5}{1024} \quad 0{,}0048828125V$$

Dijital çıkış değerlerini (10 bit'lik ADC için) Tablo-14.1'de daha iyi görebilirsiniz. 10 bit'lik ADC için, dijital değer 0000000000 (desimal 0)'dan başlar 1111111111 (desimal 1023)'e kadar gider. 0'dan başlandığı için son değer 1024 değil 1023 olur.

Analog Giriş (V)	Sayısal Bilgi Karşılığı
0,00	0000000000
0,0048828125	0000000001
0,009765625	0000000010
0,146484375	0000000011
0,01953125	0000000100
0,0244140625	0000000101
...	...
...	...
...	...
4,9951171875	1111111111

Tablo-14.1. *Maksimum 5V analog sinyalin 10 Bit'lik ADC için dijital çıkış değerleri.*

ADC biriminin analog uçlarına gelen sinyalin gerilim değerini hesaplamak içinse aşağıdaki formül kullanılır.

ADC Girişine Gelen Sinyal Gerilimi	Sayısal Çıkış Değerinin Desimal Değeri x Adım Büyüklüğü

Örnek olarak 10 bit'lik ADC ucuna gelen analog sinyalin dijital karşılığı çevrim sonucunda 1000000000 olarak bulunduysa. Bu dijital bilginin gerilim değeri verilen formüle göre aşağıda gösterildiği gibi hesaplanır.

ADC Girişine Gelen Sinyalin Gerilimi $512(\text{binary olarak } 1000000000)\text{x}0,0048828125$

$2,5\text{V}$

14.1.1. A/D Kesmesi

ADC birimi için aynı zamanda birde kesme mevcuttur. ADC işlemi bitince istenirse kesme meydana getirtilebilir. **#INT_AD** kesmesi, ADC birimi analog sinyali dijital bilgiye çevirme işlemini bitirince meydana gelir.

14.2. SETUP_ADC() FONKSİYONU

Bu fonksiyon ile A/D biriminin mod ayarı yapılır. ADC'nin kapalı olup olmayacağı, ADC frekansı olarak mikrodenetleyicinin kendi frekansını mı? yoksa dışarıdan bir giriş frekansını mı? kullanacağını, ADC işlemi için denetleyicinin kendi frekansını kullanacaksa bu frekansı kaça bölerek kullanacağı gibi işlemleri belirlemek için kullanılır. Kullanılacak "*mod* " seçenekleri mikrodenetleyici modeline göre değişebilir. Kullanılan mikrodenetleyicinin tanıtım dosyasında (16f877.h gibi) bu mod seçeneklerine bakılarak kullanılabilecek komutlar öğrenilir.

setup_adc(***mod*** **);**

setup_adc(ADC_OFF); // ADC biriminin kullanılmayacağını belirtir.
setup_adc(ADC_CLOCK_INTERNAL); // ADC çalışma frekansı
// mikrodenetleyicinin içindeki RC osilatörden olacağını belirtir.
setup_adc(ADC_CLOCK_DIV_2); //ADC çalışma frekansı
// $f_{ADC} \quad f_{osc}/2$
setup_adc(ADC_CLOCK_DIV_8); // ADC çalışma frekansı
// $f_{ADC} \quad f_{osc}/8$
setup_adc(ADC_CLOCK_DIV_32); //ADC çalışma frekansı
// $f_{ADC} \quad f_{osc}/32$

14.3. SETUP_ADC_PORTS()

ADC birimine sahip PIC mikrodenetleyicilerinde sadece A/D çevrimi için kullanılan özel giriş port'ları yoktur. Genel giriş/çıkış port'ları hem normal kullanım hem de A/D çevrimi için kullanılır. Fakat A/D çevrimi için denetleyicide hangi pin'lerin kullanılacağı bellidir. Programcının yapması gereken bu pin'lerden hangisinin A/D çevriminde analog giriş için kullanılacağı hangisinin de normal dijital giriş/çıkış işlemi için kullanılacağını derleyiciye bildirmesidir. Bu fonksiyon ile kullanılacak analog giriş pin'leri derleyiciye bildirilir. Aynı zamanda bu komut ile A/D çevriminde kullanılacak V_{REF} voltaj girişi belirlenir.

setup_adc_ports(*sabit tanım*);

Fonksiyonda "*sabit tanım*" kısmına aşağıda verilen tanımlamalar yazılır. Bu tanımlamalar kullanılan mikrodenetleyici modeline göre değişir. Bu nedenle kullanılan denetleyici tanımlama dosyasında (Bu dosyalar C:\Program Files\PICC\Devices\ dizininde bulunurlar. Örneğin 16f877.h) tüm kullanılabilir tanımlamalar incelenmelidir.

ALL_ANALOG = Denetleyicide A/D çevrimi için ayrılan tüm pin'lerin analog giriş için kullanılacağını ve V_{DD} beslemesinin $+V_{REF}$ olarak, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
NO_ANALOGS = Denetleyicide A/D çevrimi için ayrılan hiçbir pin'in analog giriş için kullanılmayacağını belirtir.
AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF = RA0, RA1, RA2, RA5, RE0, RE1, RE2 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_AN2_AN3_AN4 = RA0, RA1, RA2, RA3, RA5 girişlerinin analog giriş olacağını ve V_{DD} beslemesinin $+V_{REF}$ olarak, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_AN2_AN4_VSS_VREF = RA0, RA1, RA2, RA5 girişlerinin analog giriş olacağını ve RA3 pin'inden alınan sinyalin $+V_{REF}$ olarak kullanılacağını, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_AN3 = RA0, RA1, RA3 girişlerinin analog giriş olacağını ve V_{DD} beslemesinin $+V_{REF}$ olarak, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_VSS_VREF = RA0, RA1 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF = RA0, RA1, RA5, RE0, RE1, RE2 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, - V_{REF} olarak da RA2 pin'inden alınan sinyalin kullanılacağını belirtir.
AN0_AN1_AN2_AN3_AN4_AN5 = RA0, RA1, RA2, RA3, RA5, RE0 girişlerinin analog giriş olacağını ve V_{DD} beslemesinin $+V_{REF}$ olarak, - V_{REF} olarak da V_{SS} beslemesinin kullanılacağını belirtir.

AN0_AN1_AN2_AN4_AN5_VSS_VREF = RA0, RA1, RA2, RA5, RE0 girişlerinin analog giriş olacağını ve RA3 pin'inden alınan sinyalin $+V_{REF}$ olarak kullanılacağını, $-V_{REF}$ olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_AN1_AN4_AN5_VREF_VREF = RA0, RA1, RA5, RE0 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, $-V_{REF}$ olarak da RA2 pin'inden alınan sinyalin kullanılacağını belirtir.
AN0_AN1_AN4_VREF_VREF = RA0, RA1, RA5 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, $-V_{REF}$ olarak da RA2 pin'inden alınan sinyalin kullanılacağını belirtir.
AN0_AN1_VREF_VREF = RA0, RA1 girişlerinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, $-V_{REF}$ olarak da RA2 pin'inden alınan sinyalin kullanılacağını belirtir.
AN0 = RA0 girişinin analog giriş olacağını ve V_{DD} beslemesinin $+V_{REF}$ olarak, $-V_{REF}$ olarak da V_{SS} beslemesinin kullanılacağını belirtir.
AN0_VREF_VREF = RA0 girişinin analog giriş olacağını, $+V_{REF}$ olarak RA3 pin'inden alınan sinyalin, $-V_{REF}$ olarak da RA2 pin'inden alınan sinyalin kullanılacağını belirtir.

```
setup_adc_ports(NO_ANALOGS); // Hiçbir giriş analog değil.
setup_adc_ports(AN0_AN1_VSS_VREF); // RA1, RA2 analog, RA3 +VREF
```

14.4. SET_ADC_CHANNEL() FONKSİYONU

Analog - Dijital çevrim yapılacak kanal seçimini yapan fonksiyondur. Analog sinyali dijital sinyale çevir komutu geldiğinde mikrodenetleyicinin hangi girişteki (kanaldaki) analog sinyali dijital bilgiye çevireceğini bilmesi gerekir.

set_adc_channel(*kanal*);

Bu fonksiyonda *kanal* kısmına 0, 1, 2, ... diye kanal numaraları yazılır. Her numara kullanılan mikrodenetleyicinin bilgi kitabında (data sheet) pin diyagramı kısmında (AN0, AN1, AN2,... diye) görülebilir. Her rakam o kanalın sadece numarasını ifade eder. Yani AN2 kanalını ifade etmek için fonksiyonda *kanal* kısmına 2 yazılır.

Örneğin PIC16F877'nin pin diyagramına baktığımızda;
0= RA0/AN0 pin'ini, 1= RA1/AN1 pin'ini, 2= RA2/AN2 pin'ini, 3= RA3/AN3 pin'ini, 4= RA5/AN4 pin'ini, 5= RE0/AN5 pin'ini, 6= RE1/AN6 pin'ini, 7= RE2/AN7 pin'ini ifade eder.

```
set_adc_channel(2);   // AN2 ucundaki analog girişinin dijitale çevrileceğini
                      // bildirir.
```

Belirtilen aynı kanalda (pin'de) her okuma yapmada yeniden bu fonksiyonu kullanmaya gerek yoktur. Her okuma komutunda denetleyici en son bildirilen kanaldaki analog sinyali dijitale çevireceğini bilir. Başka bir kanaldaki (pin'deki) analog sinyali dijitale çevirmek istediğimizde tekrardan bu fonksiyonu kullanarak yeni okunacak kanalı denetleyiciye bildirmemiz gerekmektedir. **Bir ADC okuma işlemi bitmeden kanal değiştirme işlemi yapmamalıyız.** Bunun için okuma komutundan sonra, ADC çevriminin yapılabilmesi için yaklaşık 10-20µsn'lik bir gecikme vermek veya ADC çevrim işleminin bitip bitmediğini kontrol etmek gereklidir.

14.5. READ_ADC() FONKSİYONU

Bu fonksiyon belirtilen kanaldan alınan sinyalin dijital bilgi karşılığını okur ve o değerle geri döner. Önceden anlatılan "setup_adc( ), setup_adc_ports( ), set_adc_channel( )" fonksiyonlarının önceden tanımlanması gerekmektedir.

***değişken ismi* = read_adc( );**

```
setup_adc( ADC_CLOCK_INTERNAL );
setup_adc_ports( ALL_ANALOG );
set_adc_channel(1);
bilgi=read_adc( );
```

Yukarıdaki örnekte AN1 girişindeki analog sinyal dijitale çevriliyor ve "bilgi" adındaki değişkene aktarılıyor.

READ_ADC( ) fonksiyonunda, parantez içine belli *"mod"* sabitleri de yazılarak kullanılabilir.

```
read_adc( ADC_START_AND_READ);  // Normal kullanımdır. Hiçbir
// yazılmasa bu mod yazılmış gibidir. ADC işlemine başlar ve çevrimi yapar.
read_adc( ADC_START_ONLY );     // Sadece AD çevrimi işlemini yapar.
read_adc( ADC_READ_ONLY );      // Son çevrim işlemi sonucunu okur
                                // ve o değerle geri gelir.

bilgi=read_adc(ADC_READ_ONLY ); // Son ADC çevrimi sonucunu okur.
```

14.6. #DEVICE ADC=X KOMUTU

Kullanılan mikrodenetleyicinin özelliğine göre ADC modülünün kaç bit olduğu (8 bit, 10 bit, 11 bit, 16 bit) değişir. Örneğin PIC16F877 denetleyicisinde 10 bit'lik A/D modülü vardır. Fakat biz programımızda A/D biriminin 8 bit'lik olarak işlem görmesinin isteyebiliriz. Bu komut bu amaç için kullanılır. Eğer biz derleyiciye

kullanılacak A/D modülünün bit sayısını vermesek, derleyici otomatik olarak bir değer atar. Derleyiciye kullanılacak ADC'nin kaç bit'lik olacağını bildirmek için programın başına şu komutlardan biri yazılmalıdır. Bu komutlar, programa denetleyici dosyasının tanıtım komutundan (#include <16f877.h>) sonra yazılmalıdır.

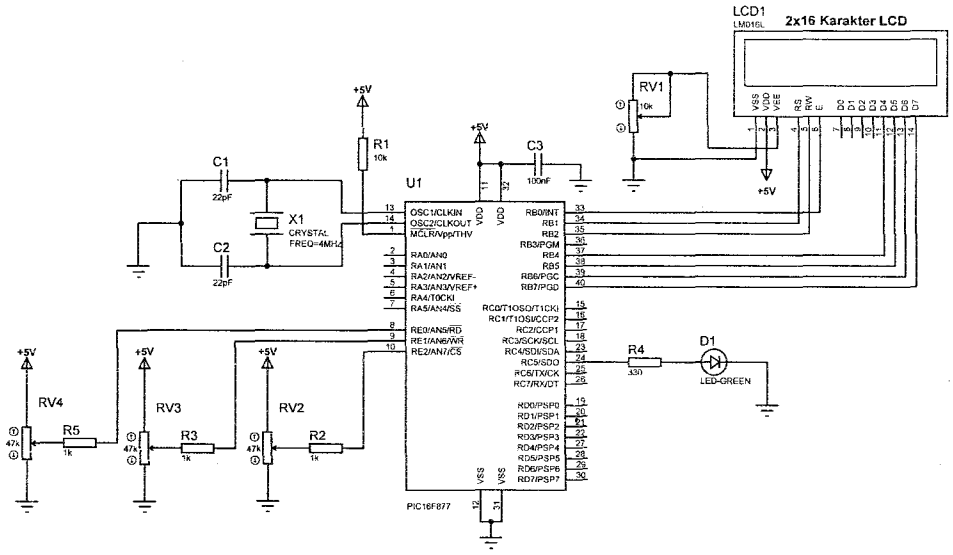
#DEVICE ADC=8 , #DEVICE ADC=10, #DEVICE ADC=11, #DEVICE ADC=16

14.7. ADC UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Uygulamada PIC16F877'nin RE0/AN5, RE1/AN6 ve RE2/AN7 uçlarına gelen analog sinyallerin dijital bilgi karşılığı sırayla ADC birimi vasıtasıyla çevrime tabi tutulmakta ve programda bu sinyallerin dijital değerleri ve gerilim değerleri hesaplanıp B port'una bağlı LCD'de görüntülenmektedir. Programda her A/D çevrim bitiminde çevriminin bittiğini belirten kesme kullanılmıştır. Bu kesmenin işlem yapıp yapmadığını görsel olarak anlamak için devrede C5 pin'ine LED bağlanmıştır. Her kesme meydana geldiğinde LED yanıp sönmektedir.

Uygulama programında LCD dosyası olarak CCS C derleyicisinin kendi "LCD.c" dosyası kullanılmıştır. Önceden de belirtildiği gibi CCS C'nin LCD.c dosyasındaki LCD'nin B port'una bağlantısı deneme kartındaki bağlantıdan farklıdır. Bu nedenle LCD'nizi deneme kartınızın LCD soketine takmamalısınız. LCD'nizi bir board üzerene yerleştiriniz ve ilgili bağlantıları deneme kartınızın sağ tarafındaki B port'u çıkış pin'lerinden zil teli kablosu alarak board üzerindeki LCD'nize bağlayınız. LCD'nin Vdd ve Vss besleme bağlantılarını da deneme kartında bulunan LCD soketinden alabilirsiniz. Port B jumper'ının LCD konumunda olması gerektiğini unutmayınız. Eğer LCD'nizi deneme kartındaki bağlantı ile kullanarak deneme kartındaki LCD soketine bağlamak istiyorsanız. Programda LCD için kullandığımız "LCD.c" dosyası yerine 13. Bölümde anlatılan Kendi LCD Dosyamızı Oluşturma veya Özel Karakterli LCD Dosyası Oluşturma kısımlarında oluşturduğumuz **"benim_LCD.c" veya "özel_LCD.c"** dosyalarını programda kullanınız.



Şekil-14.4. *ADC Uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Dahili ADC Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#device ADC=10      // 10 bit'lik ADC kullanılacağı belirtiliyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(c) //Port yönlendirme komutları C port'u için geçerli
#use fast_io(e) //Port yönlendirme komutları E port'u için geçerli

#define use_portb_lcd TRUE   // LCD B port'una bağlı

#include <lcd.c>   // lcd.c dosyası tanıtılıyor

#INT_AD                // A/D çevrimi bitti kesmesi
void ADC_Kesmesi ( )
{
   output_high(pin_c5);  // RC5 çıkışı 1
   delay_ms(200);        // 200 msn gecikme
   output_low(pin_c5);   // RC5 çıkışı 0
}
```

```
unsigned long int bilgi; // İşaretsiz 16 bit'lik tam sayı tipinde
                         // değişken tanımlanıyor
float voltaj; // ondalıklı tipte voltaj isminde değişken tanıtılıyor

//********** ANA PROGRAM FONKSİYONU*******
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_c(0x00); // C port'u komple çıkış
   set_tris_e(0x0F); // E port'u komple giriş

   output_c(0x00);   // C port'u çıkışını sıfırla

   setup_adc(adc_clock_div_32);   // ADC clock frekansı fosc/32
   setup_adc_ports(ALL_ANALOG);   // Tüm AN girişleri analog

   enable_interrupts(INT_AD);// AD çevrimi bitti kesmesi tanıtılıyor
   enable_interrupts(GLOBAL);// Aktif edilen tüm kesmelere izin ver

   lcd_init(); // LCD hazır hale getiriliyor

   printf(lcd_putc,"\f ADC UYGULAMASI ");// LCD'ye yazı yazdırılıyor
   delay_ms(1500); // 1,5 saniye gecikme veriliyor

   while(1)   // sonsuz döngü
   {

      set_adc_channel(5); // RE0/AN5 ucundaki sinyal A/D işlemine
                          // tabi tutulacak
      delay_us(20);       // Kanal seçiminde sonra bu bekleme
                          // süresi mutlaka verilmelidir
      bilgi=read_adc(); // ADC sonucu okunuyor ve bilgi değişkenine
                        // aktarılıyor
      voltaj=0.0048828125*bilgi; // Dijitale çevirme işlemine
                          // uğrayan sinyalin gerilimi hesaplanıyor
      printf(lcd_putc,"\fAN5 Kanali");
      delay_ms(1500);
      printf(lcd_putc,"\fDijital=%lu",bilgi);  //  RE0/AN5  ucundaki
                  // sinyalin dijital karşılığı LCD'ye aktarılıyor
      printf(lcd_putc,"\nVoltaj=%fV",voltaj);  //  RE0/AN5  ucundaki
                  // sinyalin gerilim değeri LCD'ye aktarılıyor
      delay_ms(2500);
      set_adc_channel(6); // RE1/AN6 ucundaki sinyal A/D işlemine
                          // tabi tutulacak
      delay_us(20);       // Kanal seçiminde sonra bu bekleme
                          // süresi verilmelidir
```

```
        bilgi=read_adc();      // ADC sonucu okunuyor ve bilgi
                               // değişkenine aktarılıyor
        voltaj=0.0048828125*bilgi;  // Dijitale çevirme işlemine
                               // uğrayan sinyalin gerilimi hesaplanıyor
        printf(lcd_putc,"\fAN6 Kanali");
        delay_ms(1500);
        printf(lcd_putc,"\fDijital=%lu",bilgi);   // RE1/AN6 ucundaki
                    // sinyalin dijital karşılığı LCD'ye aktarılıyor
        printf(lcd_putc,"\nVoltaj=%fV",voltaj);   // RE1/AN6 ucundaki
                    // sinyalin gerilim değeri LCD'ye aktarılıyor
        delay_ms(2500);

        set_adc_channel(7);    // RE2/AN7 ucundaki sinyal A/D işlemine
                               // tabi tutulacak
        delay_us(20);          // Kanal seçiminde sonra bu bekleme
                               // süresi verilmelidir
        bilgi=read_adc();     // ADC sonucu okunuyor ve bilgi
                              // değişkenine aktarılıyor
        voltaj=0.0048828125*bilgi;  // Dijitale çevirme işlemine
                              // uğrayan sinyalin gerilimi hesaplanıyor
        printf(lcd_putc,"\fAN7 Kanali");
        delay_ms(1500);
        printf(lcd_putc,"\fDijital=%lu",bilgi);   // RE2/AN7 ucundaki
                    // sinyalin dijital karşılığı LCD'ye aktarılıyor
        printf(lcd_putc,"\nVoltaj=%fV",voltaj);   // RE2/AN7 ucundaki
                    // sinyalin gerilim değeri LCD'ye aktarılıyor
        delay_ms(2500);
    }
}
/*****************************************************************
```

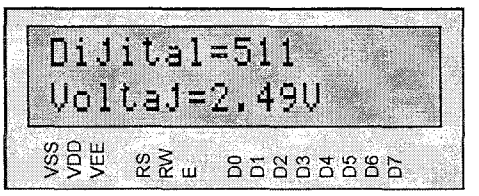
Programda `#device ADC=10` komutu ile kullanılacak dahili ADC modülünün 10 bit olacağı belirtilmiştir. Bu komut `#include <16f877.h>` komutundan sonra `#use delay (clock=4000000)` komutundan da önce kullanılmalıdır. `#use fast_io(c)` **ve** `#use fast_io(e)` **komutları ile port yönlendirmelerinin derleyici tarafından otomatik olarak yapılmayacağını programda bu işlemler için set_tris_x() komutunun kullanılacağı belirtilmiştir. LCD.c dosyasında B port'una bağlanan LCD için port yönlendirmesi yapıldığından B port'u için port yönlendirme işlemi yapılmamıştır.** Devrede LCD B port'una bağlandığı için `#define use_portb_lcd TRUE` komutu ile LCD'nin B port'una bağlandığı bildirilmiştir. Ardından da `#include <lcd.c>` komutu ile LCD.c dosyası programa eklenmiştir. `#INT_AD` ve `void ADC_Kesmesi` komutları ile A/D çevrimi bitti kesmesi fonksiyonu tanımlanmıştır. A/D çevrimi bitti kesme fonksiyonunda her A/D çevrimi sonucunda RC5 pin'ine bağlı LED'in yanıp 200 msn sonra sönmesi sağlanmıştır. Böylece A/D çevriminin yapıldığını görsel olarak görmemiz sağlanmıştır. Analog sinyalin çevrileceği 10 bit'lik dijital bilgiyi saklamak için `unsigned long int bilgi;` komutu ile işaretsiz 16 bit'lik tam sayı tipinde "bilgi" adında değişken tanımlanmıştır. 10 bit'lik ADC'de çevrim sonucu en büyük dijital olarak 1023 olur. Eğer 16 bit'lik değilde 8 bit'lik bir değişken tanımlasaydık, en fazla

255 değerini saklayabilirdik. `float voltaj;` komutu ile de analog sinyalin hesaplanan voltaj değerini saklamak için ondalıklı tipte "voltaj" isminde bir değişken tanımlanmıştır.

Ana fonksiyonun hemen altında tüm programlarda kullanmadığımız modülleri herhangi bir karışıklığa sebep vermemek için kapatıyorduk. Bu uygulamada ADC modülü kullanıldığından daha önce hep kullandığımız `setup_adc_ports(NO_ANALOGS);` ve `setup_adc(ADC_OFF);` komutları kullanılmamıştır.

`setup_adc(adc_clock_div_32);` komutu ile ADC modülünün kullanacağı frekansın osilatör frekansının 32'ye bölümü ile elde edileceği ($f_{osc}/32$) belirtilmiştir. `setup_adc_ports(ALL_ANALOG);` komutuyla da tüm ANx girişleri analog giriş olarak yönlendirilmiştir. A/D çevrimi bitti kesmesini aktif etmek için `enable_interrupts(INT_AD)` komutu kullanılmıştır. Tüm aktif kesmeler izin vermek içinde `enable_interrupts(GLOBAL);` komutu kullanılmıştır. `lcd_init();` komutu ile de LCD'nin başlangıç ayarlarının yapılması sağlanmıştır. Ardından LCD'ye `printf(lcd_putc,"\f ADC UYGULAMASI ");` komutu ile "ADC UYGULAMASI" yazısı yazdırılmıştır.

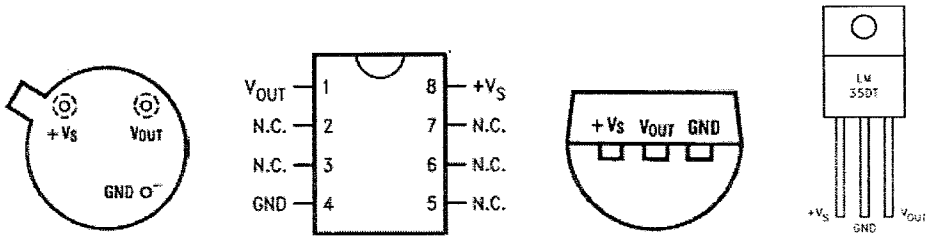
Ana fonksiyonda ilk başta `set_adc_channel(5);` komutu ile 5.kanal olan RE0/AN5 pin'indeki sinyalin A/D çevrimine tabi tutulacağı denetleyiciye bildirilmiştir. Denetleyicinin A/D kanal ayarını yapabilmesi için `delay_us(20);` komutu ile bir gecikme verilmiştir. Bu gecikme doğru işlem için mutlaka verilmelidir. Daha sonra `bilgi=read_adc();` komutundaki `read_adc()` fonksiyonu ile RE0/AN5 ucundaki sinyal A/D işlemine tabi tutturulmuştur. Elde edilen dijital sonuç ise bilgi adındaki değişkene aktarılmıştır. `voltaj=0.0048828125*bilgi;` komutu ile de 10 bit'lik ADC için hesaplanan adım büyüklüğü ($\frac{V_{REF}}{2^{10}}$ $\frac{5}{1023}$ $0,0048828125$), okunan sinyalin desimal cinsten dijital değeri ile çarpılarak RE0/AN5 pin'indeki sinyalin gerilim değeri hesaplanmış olur. `printf(lcd_putc,"\fDijital=%lu",bilgi);` komutu ile sinyalin dijital değeri ("bilgi" değişkeni) LCD'ye yazdırılmakta, `printf(lcd_putc,"\nVoltaj=%fV",voltaj);` komutuyla da LCD'de bir alt satıra geçilerek (\n komutu ile) LCD'ye sinyalin voltaj değeri yazdırılmıştır. `delay_ms(2500);` komutu ile de gecikme verilmiştir. Diğer RE1/AN6 ve RE2/AN7 pin'lerindeki analog sinyalde aynı mantıkla programda işleme tabi tutulmuştur. Sonuçta her kanaldaki sinyal bilgileri Şekil-14.5'deki gibi verilecektir.



Şekil-14.5. *ADC Uygulaması LCD görüntüsü.*

14.8. LM35 SICAKLIK SENSÖRÜ

LM35 yarı iletkeni, duyarlı bir santigrat sıcaklık sensörüdür. LM35 ölçtüğü sıcaklığı çıkışa analog sinyal olarak gönderir. Çıkışa gönderilen analog sinyal sıcaklık derecesine (^{0}C) bağlı olaral lineer bir orantı ile değişir. LM35 sensörü ek olarak harici elaman veya elemanlara ihtiyaç duymaz. LM35 sensörü -55 ile +150 ^{0}C arasında sıcaklık ölçümü yapabilmektedir. Sensörün oda sıcaklığında ±1/4 ^{0}C hata payı, tüm ölçüm aralığında ise (-55 ile +150 arası) ±3/4 ^{0}C hata payı vardır. LM35 lineer bir şekilde, **her 1 ^{0}C'lik sıcaklık artışında çıkışındaki sinyali +10 mV artırır**. LM 35 4 değişik kılıf tipinde piyasada bulunmaktadır. LM35 kılıfları Şekil-14.6'da verilmiştir. Şekilde de görüldüğü gibi LM35'lerde 3 uç bulunmaktadır. Vs ucu pozitif besleme içindirLM35 4V ile 30V arasındaki bir gerilimle çalışabilir. GND ucu şase ucudur. V_{OUT} ucu ise ölçüm yapılan sıcaklık değerinin analog sinyal olarak karşılığının verildiği çıkış ucudur.



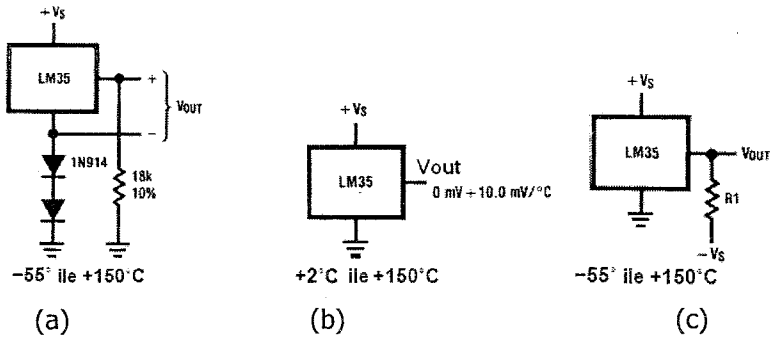
TO-46 SO-8 TO-92 TO-220

Şekil-14.6. *LM35 kılıf çeşitleri.*

LM35 sıcaklık ölçümü yapılacak yere yapıştırılabilir veya birleştirilebilir. Bu şekilde bir bağlantı gerçekleştirildiğinde ölçüm yüzeyi sıcaklığı ile LM35'in gösterdiği sıcaklık arasında 0.01 ^{0}C fark olur. Bu mantık çerçevesinde LM35'in yüzeyinin gösterdiği sıcaklık ile ortam sıcaklığı aynıdır. Fakat ortam sıcaklığı LM35'in yüzey sıcaklığından çok daha fazla veya daha az olabilir. Bu durumda gerçek sıcaklık, LM35 yüzey sıcaklığı ile ortam sıcaklığının arasındadır. Özellikle TO-92 kılıfına sahip LM35'lerde bu gibi sakıncalı durumlar ortaya çıkabilir. Çünkü devre bağlantısındaki bakır yollar ortam sıcaklığını elemanın içine taşımaktadır. Bu gibi durumlarda LM35 yapıştırıldığı yüzey sıcaklığını ölçmek yerine ortam sıcaklığından çok fazla etkilenmiş yanlış bir ölçme yapmış olacaktır. Bahsedilen etkilenmeyi en aza indirmek için kabloları dış sıcaklıktan etkilenmeyecek şekilde epoksi gibi maddelerle kaplamaktır. TO-46 kılıfına sahip LM35'ler metal yüzeylere lehimlenebilir. LM35 aynı zamanda Fahrenhayt olarak sıcaklık ölçümü içinde kullanılabilir. Bunun için gerekli bağlantılar için elemanın katalog bilgilerine bakabilirsiniz.

Kullanım şartları, sıcaklık ölçüm aralığı gibi durumlara göre LM35'i değişik bağlantılarda kullanmak gereklidir. Şekil-14.7'de bazı bağlantı şekilleri verilmiştir. Ayrıntılı bilgi için elemanın bilgi sayfalarına (datasheet) bakınız. **LM35'in -55 ile**

+150 ^{0}C aralığında kullanımında çıkış voltaj aralığı -1.0V ile +6V arasındadır.



(a) (b) (c)

Şekil-14.7. *LM35 tipik uygulama bağlantıları.*

Şekil-14.7-a'da bağlantı kablosu kapasitesi etkisi giderilmiş -55^{0}C ile +150^{0}C arasında ölçüm yapılacak bağlantı görülmektedir. Şekil-14.7-b'de ise LM35'in sadece +2^{0}C ile +150^{0}C arasında pozitif sıcaklık değerlerini belirleyecek çıkış verdiği bağlantı görülmektedir. Bu bağlantıda çıkış 0mV iken ölçülen sıcaklık +2^{0}C'dir. Sıcaklık 3^{0}C'de olduğunda çıkış 10mV olur. Örneğin 20^{0}C'de çıkış voltajı 180 mV'tur. Şekil-14.7-c'deki bağlantı ise -55^{0}C ile +150^{0}C arasında ölçüm yapabilir. Bu bağlantıda iki kaynak kullanılmalıdır. Bağlantıdaki direnç değeri R1=-V_S/50µA formülü ile hesaplanır.

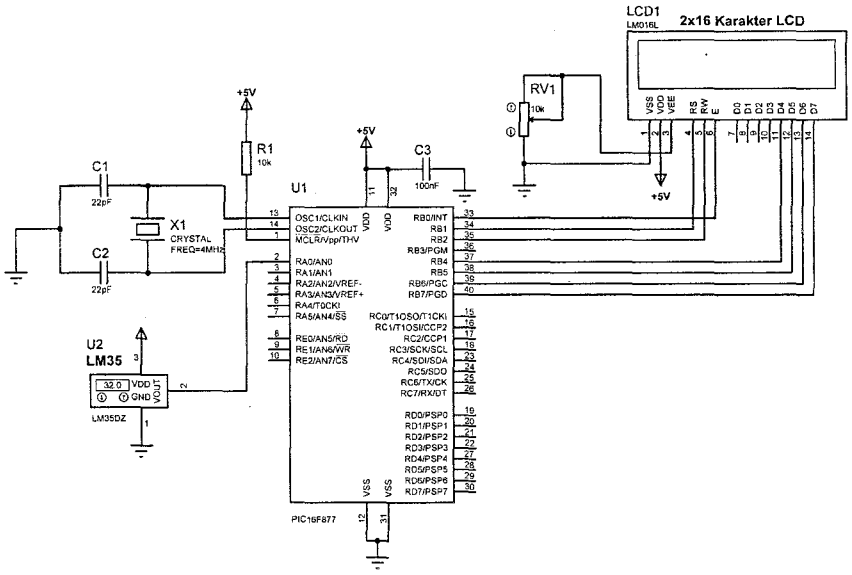
14.9. LM35 SICAKLIK SENSÖRÜ UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada LM35 sıcaklık sensörü denetleyicinin RA0/AN0 ucuna bağlanmıştır. LM35 bağlantısı olarak da Şekil-14.7-b'de verilen +2^{0}C ve 150^{0}C arasında ölçüm yapan bağlantı şekli kullanılmıştır. LM35'den alınan sıcaklık karşılığı analog sinyaller, denetleyicinin dahili 10 bit'lik ADC birimi vasıtası ile dijitale çevrilmiştir. A/D çevrimi için referans gerilimi V_{DD} yani 5V seçilmiştir. Çözünürlük olarak, $5V/2^{10}$=0.0048828125 Volt ≈ 4,88 mV'luk bir hassasiyetle analog sinyal örneklenebilmektedir. Bu da her 1 ^{0}C'de 10mV çıkış veren LM35 için, 0.5^{0}C aralıkta ölçüm yapmamıza olanak sağlar. Program vasıtası ile de bu değerlerin karşılığı olan sıcaklık bilgileri LCD'de gösterilmiştir.

Devreyi PIC PROG/DEKA PIC16F87x deneme kartında denerken LM35 elemanınızı bir board üzerine yerleştiriniz. Gerekli V_{DD} ve V_{SS} gerilimlerini deneme kartının sol tarafında bulunan konektörden zil teli ile alınız. LM35'in çıkış ucundan aldığınız zil teli kablosunu da yine deneme kartının sol tarafında bulunan konektördeki AN0 ucuna yerleştiriniz. **GND bağlantısını LCD'ye bağladığınız şase ucundan almanız yanlış ölçüm yapmanıza neden olabilir.**

Uygulama programında LCD dosyası olarak CCS C derleyicisinin kendi "LCD.c" dosyası kullanılmıştır. Önceden de belirtildiği gibi CCS C'nin LCD.c dosyasındaki LCD'nin B port'una bağlantısı deneme kartındaki bağlantıdan farklıdır. Bu nedenle LCD'nizi deneme kartınızın LCD soketine takmamalısınız. LCD'nizi bir board üzerene yerleştiriniz ve ilgili bağlantıları deneme kartınızın sağ tarafındaki B port'u çıkış pin'lerinden zil teli kablosu alarak board üzerindeki LCD'nize bağlayınız. LCD'nin Vdd ve Vss besleme bağlantılarını da deneme kartında bulunan LCD soketinden alabilirsiniz. Port B jumper'ının LCD konumunda olması gerektiğini unutmayınız. Eğer LCD'nizi deneme kartındaki bağlantı ile kullanarak deneme kartındaki LCD soketine bağlamak istiyorsanız. Programda LCD için kullandığımız "LCD.c" dosyası yerine 13. Bölümde anlatılan Kendi LCD Dosyamızı Oluşturma veya Özel Karakterli LCD Dosyası Oluşturma kısımlarında oluşturduğumuz **"benim_LCD.c" veya "özel_LCD.c"** dosyalarını programda kullanınız.



Şekil-14.8. *LM35 Sıcaklık sensörü uygulaması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile LM35 Sıcaklık Sensörü Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#device ADC=10  // 10 bit'lik ADC kullanılacağı belirtiliyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
```

```
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli

#define use_portb_lcd TRUE   // LCD B port'una bağlı

#include <lcd.c>   // lcd.c dosyası tanıtılıyor

unsigned long int bilgi; // İşaretsiz 16 bit'lik tam sayı tipinde
                            // değişken tanımlanıyor
float voltaj, sicaklik;  // ondalıklı tipte değişkenler tanıtılıyor

//********** ANA PROGRAM FONKSİYONU*******
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x01);  // RA0 Giriş olarak yönlendiriliyor

   setup_adc(adc_clock_div_32);   // ADC clock frekansı fosc/32
   setup_adc_ports(AN0);          //RA0/AN0 girişi analog
   lcd_init();   // LCD hazır hale getiriliyor

   set_adc_channel(0); // RA0/AN0 ucundaki sinyal A/D işlemine tabi
                       // tutulacak
   delay_us(20);       // Kanal seçiminde sonra bu bekleme süresi
                       // verilmelidir
   printf(lcd_putc,"\fSicaklik="); // LCD'ye yazı yazdırılıyor

   while(1)   // sonsuz döngü
   {
      bilgi=read_adc(); // ADC sonucu okunuyor ve bilgi değişkenine
                        // aktarılıyor
      voltaj=(0.0048828125*bilgi)*1000; // Dijitale çevirme işlemine
        // uğrayan sinyalin mV cinsinden gerilim değeri hesaplanıyor
      sicaklik=(voltaj/10)+2;    // Her 10mV'ta 1 derece artma

      lcd_gotoxy(10,1); // İmleç 1. satır 10.sütunda
      printf(lcd_putc,"%5.1f'C",sicaklik); // Sıcaklık değeri LCD'de
                                           // gösteriliyor.
// Gösterilecek ondalıklı değerin toplam basamak sayısı 5 olacak.
// Gösterilecek değerin ondalıklı kısmı ise 1 basamak olacak. %5.1f
// komutu ile bu işlemler yaptırılmıştır.
      delay_ms(100); // 100 msn gecikme
   }
}
//*****************************************************************
```

Programda `setup_adc_ports(AN0);` komutu ile RA0/AN0 girişinin analog giriş olarak kullanılması sağlanmıştır. Bu komutla aynı zamanda $+V_{REF}$ olarak PIC beslemesi olan V_{dd}'nin kullanılacağını, $-V_{REF}$ olarak da V_{SS} ucunun kullanılacağı belirtilmiştir. ADC modülünün frekansı olarak da `setup_adc(adc_clock_div_32);` komutu ile fosc/32 değeri belirlenmiştir. `set_adc_channel(0);` komutu ile AN0 girişindeki (kanalındaki) sinyalin A/D çevrimine tabi tutulacağı derleyiciye bildirilmiştir. Kanal seçiminden sonra, bu işlemin denetleyici tarafından yapılabilmesi için gerekli süre `delay_us(20);` komutu ile verilmiştir. Sonsuz döngüden önce `printf(lcd_putc,"\fSicaklik=");` komutu ile LCD temizlenmiş ve LCD'ye "Sicaklik=" ifadesi yazdırılmıştır. Sonsuz döngüde `(while(1)) bilgi=read_adc();` komutu ile AN0 girişindeki sinyal dijital bilgiye çevrilmiş ve "bilgi" değişkenine atanmıştır. daha sonra `voltaj=(0.0048828125*bilgi)*1000;` komutu ile elde edilen dijital bilginin gerilim değeri milivolt (mV) olarak hesaplanmıştır. Formülde 10 bit'lik bir ADC'nin adım değeri sinyalin dönüştürülen dijital değeri ile çarpılmıştır. Bu değerde 1000 ile çarpılarak sonucun Volt değeri, miliVolt karşılığına çevrilmiştir. Elde edilen bu gerilim değerinde her 10mV, 1^0C'ye karşılık geldiğinden voltaj/10 formülü ile ölçülen gerilim değerinin kaç dereceye karşılık geldiği bulunur. Fakat uygulamamızda LM35 bağlantısı önceden belirtildiği gibi $+2^0$C ve 150^0C arasında ölçüm yapacak şekilde olduğundan `sicaklik=(voltaj/10)+2;` komutunda olduğu gibi formüle +2 değeri eklenmiştir. Çünkü kullandığımız bağlantıda 2^0C'de çıkış 0mV'tur. Ölçüm sonucunun gösteriminde yanlışlık olmasın diye bu değer formüle eklenmiştir.

Sonsuz döngüden önce `printf(lcd_putc,"\fSicaklik=");` komutu ile LCD temizlenmiş ve LCD'ye "Sicaklik=" ifadesi yazdırılmıştı. `lcd_gotoxy(10,1);` komutu ile 1. satır 10. sütuna gidilmiş ve `printf (lcd_putc, "%5.1f'C", sicaklik);` komutu ile ölçülen sıcaklık değeri LCD'ye yazdırılmıştır. Bu sayede sonsuz döngüde LCD her zaman silinip LCD'ye "Sicaklik=" ifadesi yazdırılmasına gerek kalmamıştır. Bu ifade ekranda sabit kalmakta, sadece ölçülen sıcaklık değeri değişmektedir.

`printf(lcd_putc,"%5.1f'C",sicaklik);` komutunda `%f` ifadesi bilindiği gibi buraya değişken yazdırılacağını ve bu değişkenin ondalıklı tipte olacağını (f kısmı) ifade ediyordu. Yazdırılacak değişken ismi de, komutta en son kullanılan çift tırnaktan sonra virgül verildikten sonra yazılıyordu. Bu komutta da böyle yapıldı. Fakat daha önceden görmediğiniz bir şekilde `%f` yerine, `%5.1f` yazılmıştır. Bunun anlamı, yazılacak değişkenin toplam uzunluğu 5 karakter (sayıdaki ondalık noktası dâhil), ondalık kısmının uzunluğu ise 1 karakterli olacak anlamındadır.

15. Bölüm

DÂHİLİ EEPROM İŞLEMLERİ

- Dâhili EEPROM
- Flash Program Hafızası
- EEPROM Kesmesi (#INT_EEPROM)
- READ_EEPROM() Fonksiyonu
- WRITE_EEPROM() Fonksiyonu
- READ_PROGRAM_EEPROM() Fonksiyonu
- WRITE_PROGRAM_EEPROM() Fonksiyonu
- ERASE_PROGRAM_EEPROM() Fonksiyonu
- Dâhili EEPROM Uygulaması (Hafızalı Kronometre)
- Flash Program Hafızası Uygulaması

Bu bölümde PIC16F877 denetleyicisi içinde bulunan dâhili EEPROM birimi tanıtılmıştır. Ardından CCS C derleyicisinde dâhili EEPROM birimi ile ilgili fonksiyonlar ve görevleri anlatılmıştır. Dâhili EEPROM uygulaması ile de bilgiler pratik olarak desteklenmiştir.

15.1. DÂHİLİ EEPROM

PIC16F877 denetleyicisi **dâhili data EEPROM** ve **Flash Program hafızası** alanı içerir. EEPROM'a yazılan bilgiler PIC'in enerjisi kesilse dahi silinmezler. Ancak programla bu belleklere bilgi yazılabilir, okunabilir ve silinebilir. Dâhili EEPROM'a 8 bit'lik veriler yazılabilir ve dâhili EEPROM adreslerindeki 8 bit'lik veriler okunabilir. PIC16F877 denetleyicisi 2 Kb'lık data EEPROM hafızasına sahiptir. Yani 256 tane 8 bit'lik veri, dâhili veri EEPROM birimine kaydedilebilir. PIC16F877 için Data EEPROM adresi 0x00'dan 0xFF'e kadardır. Dâhili EEPROM birimini kontrol eden kaydedici **EECON1** ve **EECON2** kaydedicileridir. EECON1 kaydedicisinin bit'leri ve açıklamaları aşağıda verilmiştir. EECON2 kaydedicisi fiziksel olarak yoktur. EEPROM'da ulaşılmak istenen adres bilgileri **EEADR** ve **EEADRH** kaydedicilerinde tutulur. Adres bilgisi 13 bit'ten ibarettir. Adres bilgisinin yüksek değerlikli 8 bit'ini (MSB kısmını) EEADRH kaydedicisi, düşük değerlikli 5 bit'ini (LSB kısmını) EEADR kaydedicisi tutar. EEPROM birimindeki okuma-yazma işlemlerindeki veri ise

EEDATA ve **EEDATH** kaydedicilerinde tutulur. Dâhili EEPROM için sadece EEDATA kaydedicisi yeterlidir. Fakat EEPROM data hafızasına 8 bit'lik veriler yazılırken, flash program hafızasına 14 bit'lik veriler yazılır. Bu nedenle flash programa veri yazılırken verinin yüksek değerlikli 8 bit'ini EEDATH kaydedicisi, düşük değerlikli 6 bit'ini de EEDATA kaydedicisi tutar.

EECON1 KAYDEDİCİSİ

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

Şekil-15.1. *EECON1 kaydedicisi.*

Bit 7 **EEPGD** = Program veya Data EEPROM seçme bit'i
1: Program EEPROM
0: Data EEPROM

Bit 6-4 = Kullanılmayan bit'ler. Okunduğunda "0" değeri alınır.

Bit 3 **WRERR** = EEPROM hata bayrağı bit'i
1: EEPROM yazma işlemi vakitsizce son buldu.
0: EEPROM yazma işlemi tamamlandı

Bit 2 **WREN** = EEPROM yazma işlemi etkinleştirme bit'i
1: EEPROM yazma çevrim işlemine izin verir
0: EEPROM yazma işlemine izin verme

Bit 1 **WR** = Yazma işlemi kontrol bit'i
1: Yazma işlemini başlat. Yazma işlemi bitince otomatik olarak lojik-0 olur.
0: EEPROM'a yazma döngüsü tamamlandı

Bit 0 **RD** = Okuma işlemi kontrol bit'i
1: EEPROM okuma işlemini başlatır.Okuma işlemi bitince otomatik olarak lojik-0 olur.
0: EEPROM okuma işlemine izin verme

15.1.1. EEPROM Kesmesi (#INT_EEPROM)

EEPROM yazma işlemi bitince istenirse INT_EEPROM kesmesi meydana gelir. Bu kesmeyi kullanarak EEPROM yazma işleminin bittiğinin anlaşılabildiği gibi yazma işleminin bitiminde uygulamanızda yaptırmak istediğiniz işlemleri gerçekleştirebilirsiniz. EEPROM kesmesi #int_eeprom olarak ifade edilir.

15.2. DÂHİLİ EEPROM OKUMA-YAZMA FONKSİYONLARI

Dâhili EEPROM kontrol kaydedicisi EECON1 kaydedicisini kontrol etmek için CCS C çok kolay kullanımlı fonksiyonlara sahiptir.

15.2.1. READ_EEPROM() Fonksiyonu

Belirtilen data EEPROM adresindeki bilgiyi okuma işlemi yapan ve okuduğu bilgi ile geri dönen bir fonksiyondur. Fonksiyondaki *"adres"* kısmına kullanılan denetleyicinin modeline göre 8 veya 16 bit'lik tamsayı tipinde bir adres değeri yazılır. Bu adres değeri 0'dan başlar ve kullanılan denetleyicinin kapasitesine bağlı olarak son sınıra kadar değer yazılabilir.

***değişken ismi* = read_eeprom(*adres*);**

```
bilgi = read_eeprom( 0 );  // 0 numaralı adresteki veriyi okur.
i = read_eeprom( 5 );      // 5 numaralı adresteki veriyi okur.
```

15.2.2. WRITE_EEPROM() Fonksiyonu

Bu fonksiyon dahili data EEPROM'da istenen adrese istenen veriyi (bilgiyi) yazma işlemini yapar. Fonksiyondaki *"adres"* kısmına kullanılan denetleyicinin modeline göre 8 veya 16 bit'lik tam sayı tipinde bir adres değeri yazılabilir. Fonksiyondaki *"bilgi"* kısmına ise 8 bit'lik bir tamsayı değeri yazılabilir. EEPROM yazma fonksiyonu birkaç mili saniyede yazma işlemini gerçekleştirir.

write_eeprom(*adres, bilgi*);

```
write_eeprom(25, 15);  // 25 numaralı adrese 15 bilgisini yazar.
x=read_eeprom(25);     // x değişkeni şimdi 15 değerine eşit olur.
```

15.3. FLASH PROGRAM BELLEĞİ OKUMA-YAZMA İŞLEMLERİ

PIC denetleyicisi içine yüklenen program kodları **flash program belleğinde** saklanır. Denetleyici işlemleri bu bellekteki kodlara göre yapar. 14 bit'lik PIC denetleyicilerde program kodları 14 bit'lik verilerden oluşur. PIC16F877 denetleyicisi 8K'lık program belleğine sahiptir ve program belleği adresi 0x0000 ile 0x1FFF arasındadır. Bu flash program belleğine programlama kartlarıyla programlar yazılmaktadır. Aynı belleklere denetleyici işlem yaparken de erişilebilir. Yani denetleyici normal işlemlerini yaparken program kodları değiştirilebilir, silinebilir veya okunabilir. İşte bu işlemler yapmak için CCS C'de 5 fonksiyon vardır.

15.3.1. READ_PROGRAM_EEPROM() Fonksiyonu

Flash program EEPROM'unda istenen bir adresteki veriyi okuma işlemini yapar. Bu fonksiyonun kullanılabilmesi için kullanılan mikrodenetleyicinin program hafızasını okumaya izin veren özellikte olması gerekmektedir. Fonksiyonda *"adres"* kısmı 14

bit'lik işlem koduna (opcode) sahip mikrodenetleyicilerde 16 bit, 16 bit'lik işlem koduna sahip mikrodenetleyicilerde ise 32 bit'lik bir değerdir. Fonksiyondaki *"değişken"* kısmı ise 16 bit'lik bir değişken olmalıdır.

***değişken ismi* = read_program_eeprom(*adres*);**

```
i = read_program_eeprom (34);  // Program hafızasında 0x34 adresindeki
                               // komut değeri okunuyor ve i değişkenine aktarılıyor
```

15.3.2. WRITE_PROGRAM_EEPROM() Fonksiyonu

Flash program EEPROM'daki istenen bir adrese istenen bir bilgiyi yazma işlemini yapar. Bu fonksiyonun kullanılabilmesi için kullanılan mikrodenetleyicinin program hafızasına yazmaya izin veren özellikte olması gerekmektedir. Fonksiyonda *"adres"* kısmı 14 bit'lik işlem koduna (opcode) sahip mikrodenetleyicilerde 16 bit, 16 bit'lik işlem koduna sahip mikrodenetleyicilerde ise 32 bit'lik bir değerdir. Fonksiyondaki *"bilgi"* kısmı ise 16 bit'liktir.

write_program_eeprom(*adres, bilgi*);

```
write_program_eeprom( 3, 0x1300); // Program belleğindeki 3 numaralı
                                  // adrese 1300 değerini yazar.
```

15.3.3. ERASE_PROGRAM_EEPROM() Fonksiyonu

Program hafızasındaki istenen adresteki bilgiyi siler. Fonksiyonda *"adres"* kısmı 14 bit'lik işlem koduna (opcode) sahip mikrodenetleyicilerde 16 bit, 16 bit'lik işlem koduna sahip mikrodenetleyicilerde ise 32 bit'lik bir değerdir.

erase_program_memory(*adres*);

```
erase_program_memory(0x1000); // 1000 adresindeki bilgiyi siler
```

15.4. DATA EEPROM VE FLASH PROGRAM HAFIZASI KORUMA İŞLEMLERİ

PIC16F877 denetleyicisinde 2 tip koruma vardır. Birincisi data EEPROM hafızasını koruma diğeri de flash program hafızasını koruma. Data EEPROM korumasının aktif olup olmamasına bakılmaksızın data EEPROM içeriği program işlerken okunup yazılabilir. Data EEPROM koruması aktif ise dışarıdan cihazların data EEPROM içeriğine ulaşması engellenir. Data EEPROM korumasının programın başında konfigürasyon komutları ile belirtildiğini hatırlayınız. #fuse CPD ifadesi data

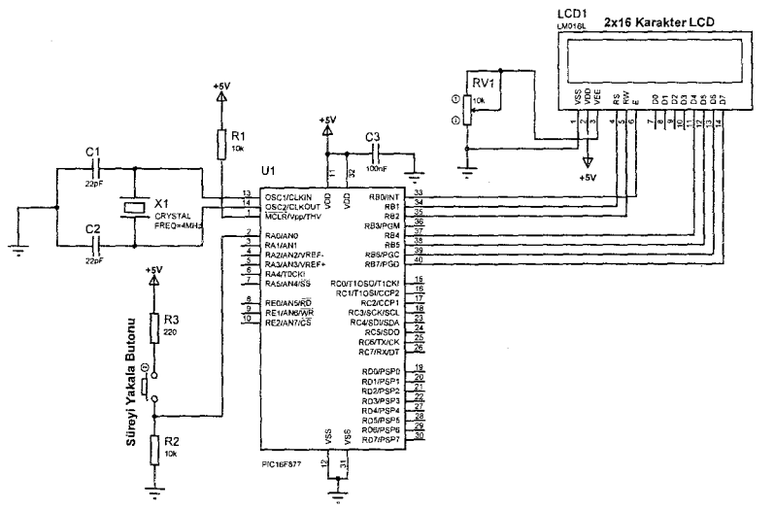
EEPROM kod korumasını aktif yaparken, #fuse NOCPD ifadesi data EEPROM kod korumasını kapatır.

Flash program hafızasını da korumalı yapabilirsiniz. Özelikle ticari uygulamalarda bu işlem çok kullanılmaktadır. Program hafızası korumaya alındığında bir daha denetleyiciye yüklenen program, programı yükleyen dâhil kimse tarafından okunamaz. #fuse NOPROTECT komutu ile program kod hafızası koruması pasif hale getirilir. #fuse PROTECT komutu ile de program kod hafızasının okunması önlenmiş olunur. Program hafızasının tümü okumaya karşı korumalı yapılabildiği gibi, sadece belli bir kısmı da okumaya karşı korunabilir. Bu işlem içinde #fuse PROTECT_5% ve #fuse PROTECT_50% komutları kullanılır. Program hafızasına yazma işlemini korumaya almak için #fuse WRT komutu, korumayı pasif etmek için #fuse NOWRT komutu kullanılır.

15.5. DÂHİLİ EEPROM UYGULAMASI (Hafızalı Kronometre)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada basit bir hafızalı kronometre (süre ölçer) uygulaması gerçekleştirilmiştir. B port'una bağlanan LCD'nin ilk satırında 0'dan 255'e kadar artan süre değerleri saniye ve mili saniye olarak görülmektedir. RA0 girişine bağlanan butona basılınca LCD'nin alt satırına o anki süre değeri yazdırılmaktadır. Aynı zamanda her işlemde yakalanan süre değeri data EEPROM'a kaydedilmektedir. Böylece bir güç kaynağı kesintisinde veya arızada yakalanan değer korunmuş olmaktadır. Uygulamanızı denerken PIC'i resetleyip tekrar çalıştırdığınızda LCD ekranında resetlemeden önceki yakalanan süre değerinin korunduğunu göreceksiniz. Süre hesaplaması için Timer0 zamanlayıcı olarak kullanılmıştır. Timer0 yaklaşık her 10 msn'de bir kesme oluşturacak şekilde ayarlanmıştır. Kesme fonksiyonunda da _10ms isimli milisaniye değerini tutan değişken her kesmede 1 arttırılmaktadır. _10ms değeri 100 olunca da (100 msn=1saniye), saniye değeri 1 arttırılmaktadır.



Şekil-15.2. *Dahili EEPROM uygulaması devre şeması.*

Devre programı;

```
/*********************************************************************
      PIC16F877 ile Dahili EEPROM Uygulaması(Hafızalı Kronometre)
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*********************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli

#define use_portb_lcd TRUE   // LCD B port'una bağlı

#include <lcd.c> // LCD.c dosyası programa ekleniyor

int _10ms=0;            // Tam sayı tipinde değişken tanımlanıyor
unsigned int saniye=0; // İşaretsiz tam sayı tipinde değişken

//****************** Timer0 Kesmesi ****************************
#int_timer0  // Timer0 kesmesi. Her 10 msn'de bir kesme oluşur.
void  timer0_kesme ()    // Kesme fonksiyonu ismi
{
    set_timer0(217); // TMR0 değeri belirleniyor

    _10ms++;          // _10ms değişkeni değerini 1 arttır
    if (_10ms==100) //_10ms değeri 100 olunca. (1 saniye olunca)
    {
```

```
        _10ms=0;          // _10ms değişkenini sıfırla
        saniye++;         // saniye değişkenini bir arttır
        if (saniye==255) // Eğer saniye değişkeni 999 olursa sıfırla
           saniye=0;      // saniye değişkenini sıfırla
     }
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x01);    // RA1 pin'i giriş

   lcd_init();          // LCD kurulumu yapılıyor

   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256); // Timer0 ayarları
                                                 // yapılıyor
   set_timer0(217);    // TMR0 değeri belirleniyor

   enable_interrupts(INT_timer0);// int_timer0 kesmesini aktif yapar
   enable_interrupts(GLOBAL);     // Aktif edilen kesmelere izin ver

   lcd_gotoxy(1,1);                      // İmleç 1.sütun, 1. satırda
   printf(lcd_putc,"Sure=%03u",saniye);// Süre LCD'ye yazdırılıyor
   lcd_gotoxy(1,2);                      // İmleç 1.sütun, 2. satırda
   printf(lcd_putc,"Sonuc=%03u",read_eeprom(0));//EEPROM'da
         // 0.adreste saklanan süre okunuyor ve LCD'ye yazdırılıyor

   while(1)       // Sonsuz döngü
   {
      if (input(pin_a0))  // RA0 girişindeki butona basılmışsa
      {
         delay_ms(20);     // Buton arkını önlemek için
         write_eeprom(0,saniye); //0x00 adresine saniye değerini yaz
         lcd_gotoxy(7,2);         // İmleç 7.sütun, 2. satırda
         printf(lcd_putc,"%03u",saniye);//LCD'ye saniye değerini yaz
      }
      lcd_gotoxy(6,1);                  // İmleç 6.sütun, 1. satırda
      printf(lcd_putc,"%03u",saniye); // Süre LCD'ye yazdırılıyor
      lcd_gotoxy(9,1);                  // İmleç 9.sütun, 1. satırda
      printf(lcd_putc,".%02d",_10ms); // msn değerini LCD'ye yaz
   }
}
//*****************************************************************
```

Programda Timer0 zamanlayıcı olarak kullanılmıştır. `setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256);` komutu ile Timer0'ın denetleyicinin kendi frekansından besleneceği ve bölme oranının 256 olduğu belirtilmiştir. `set_timer0(217);` komutu ile de TMR0 kaydedicisi değeri 217 olarak belirtilmiştir. Bu şartlarda Timer0 kesmesi süresi yaklaşık 10 msn bulunur.(Kullanılan osilatör frekansı 4Mhz olduğundan ve PIC osilatör frekansını 4'e bölüp kullandığından f_{osc}=4Mhz/4=1Mhz olduğunu hatırlayınız)

Kesme Süresi T_{osc}x(Bölme Oranı)x(256 TMR0'a yazılan değer)

$$= \frac{1}{f_{osc}} \text{x256x(256-217)} = \frac{1}{1\text{Mhz}} \text{x256x39}$$

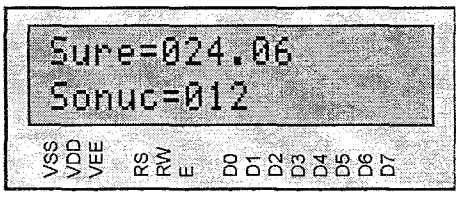
=1µsnx9984=9984µsn 10msn

Kesme fonksiyonunda, her kesmede yani her 10msn'de "_10ms" değişkeni 1 arttırılmıştır. "_10ms" değişkeni 100 olunca ise yani toplam 10msn*100=1000msn=1saniye geçince "saniye" değişkeni 1 arttırılmaktadır. Kesme fonksiyonunda ayrıca saniye değeri 255 olunca sıfırlanmaktadır.

Sonsuz döngüden önce `lcd_gotoxy(1,1);` komutu ile imleç 1. sütun 1.satıra alınmış ve `printf(lcd_putc,"Sure=%03u",saniye);` komutu ile ekrana süre değeri yazdırılmıştır. Komuta dikkat edilirse %03u kısmı ile buraya "saniye" değişkeninin geleceği, bu değişkenin işaretsiz (unsigned) formatta (komuttaki u ile), 3 basamaklı olarak (komuttaki 3 ile) ve boşta kalan basamaklara 0 yazılacak (komuttaki 0 ile) şekilde LCD'ye yazdırılacağı belirtilmiştir. Bu sayede örneğin süre değeri 5 ise LCD'de 005 şeklinde görülecektir. `printf(lcd_putc,"Sonuc=%03u",read_eeprom(0));` komutuyla ilk LCD'ye EEPROM hafızasında tutulan değerin yazdırılması sağlanmıştır. Komutta %03u kısmı ile buraya bir değişken yazdırılacağı belirtilmiştir. Buraya yazdırılacak değişken ise read_eeprom(0) komutu ile data EEPROM hafızasında 0. adresten okunan değer olacaktır.

Sonsuz döngüde `(while(1)) if (input(pin_a0))` komutu ile RA0 girişine bağlı butona basılıp basılmadığı kontrol edilmektedir. Eğer butona basılmışsa if döngüsü işlem görecektir. Döngüde `write_eeprom(0,saniye);` komutu ile data EEPROM'da 0. adrese "saniye" değişkeni içeriği kaydedilmektedir. Ardından lcd_gotoxy(7,2); komutu ile imleç 7. sütun 2.satıra alınmakta ve `printf(lcd_putc,"%03u",saniye);` komutuyla LCD'de 2.satıra yakalanan süre değeri yazdırılmaktadır. Böylece butona basıldığı andaki süre değeri hem data EEPROM'a kaydedilmekte hem de LCD'de gösterilmektedir.

Sonsuz döngüde sürekli olarak "saniye" değişkeni ile "_ms10" değişkeni değerleri LCD'ye aktarılmaktadır. Sonuçta LCD ekranında Şekil-15.3'teki gibi bir görüntü oluşur.



Şekil-15.3. *Dahili EEPROM (Hafızalı Kronometre) uygulaması LCD görüntüsü.*

15.6. FLASH PROGRAM HAFIZASI UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada Flash program hafızasında bulunan bilgiler LCD'de adres ve içindeki değer olarak gösterilmiştir. Ayrıca RA0 girişine bağlanan butona basılınca LCD'de an görünen program adresi içeriğinin sıfırlanması sağlanmıştır. Buton ile program kodları sıfırlandığında PIC bir daha düzgün çalışmayacaktır. Yeniden düzgün çalışmak için PIC'e yeniden programı yüklemeniz gerekmektedir. Uygulama devresi bir önceki uygulama devresi ile aynıdır (Şekil-15.2).

Devre programı;

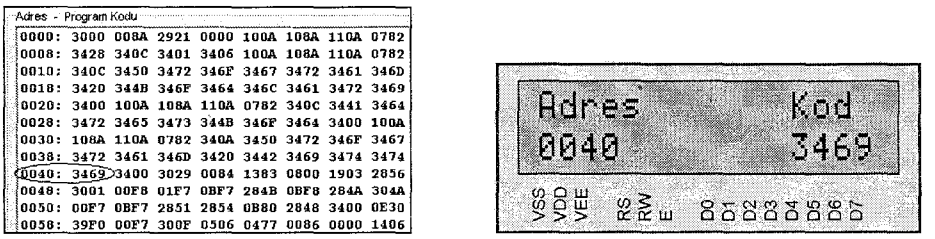
```
/*******************************************************************
      PIC16F877 ile Flash Program Hafızası Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*******************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           //osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli

#define use_portb_lcd TRUE    // LCD B port'una bağlı

#include <lcd.c> // LCD.c dosyası programa ekleniyor

int16  adres=0,deger=0;

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
```

```
   setup_adc(ADC_OFF);                // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);               // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);               // CCP2 birimi devre dışı

   set_tris_a(0x01);    // RA1 pin'i giriş

   lcd_init();          // LCD kurulumu yapılıyor

   printf(lcd_putc,"\fProgram Kodlari"); // LCD'ye yazı yazdırılıyor
   delay_ms(1000);
   printf(lcd_putc,"\fAdres       Kod"); // LCD'ye yazı yazdırılıyor

   while  (!(adres==0x3FFF))  //Program kodu 0x3FFF değerine eşit
//oluncaya kadar döngüyü işle. 0x3FFF boş olan program bellek kodu.
   {
      deger=read_program_eeprom(adres); // Program kod hafızasında
belirtilen adresteki kodu oku
      lcd_gotoxy(1,2); // İmleç 1.sütun 2.satırda
      printf(lcd_putc,"\n%lx        %lx",adres,deger); // LCD'ye
            // program kod hafızası adresini ve içindeki kodu yaz
      delay_ms(500);
      adres++; // adres değişkeni içeriğini 1 arttır

      if (input(pin_a0)) // Eğer RA0 girişindeki butona basılırsa
      {
         delay_ms(20);  // Buton arkını önlemek için
         write_program_eeprom(adres, 0x0000); // Belirtilen program
                                  // kod hafızası adresine 0 bilgi yaz
      }
   }
   printf(lcd_putc,"\fProgram Bitti..."); // Program kod hafızasının
                                          //sonuna geldiğini bildirir
}
//*****************************************************************
```

Programda program kod hafızasının adres değerini tutacak değişken "adres" ile o adresteki bilgiyi tutacak "deger" değişkenleri `int16 adres=0,deger=0;` komutlarıyla 16 bit'lik olarak tanımlanmıştırlar. Çünkü program kod belleği 14 bit'liktir. PIC program hafızasında boş olan alanların değeri 0x3FFF olduğundan, `while (!(adres==0x3FFF))` komutu ile adres değeri bu değere eşit olmadığı sürece while döngüsünün işlemesi sağlanmıştır. Bilindiği gibi komuttaki "!" işareti "mantıksal değil" anlamındadır. `deger=read_program_eeprom(adres);` komutu ile program hafızasındaki "adres" değişkeni ile belirtilen adreste bulunan kod okunarak "deger" değişkenine aktarılmıştır. Ardından `printf(lcd_putc,"\n%lx %lx",adres,deger);` komutu ile de hem adres, hem de kod değerleri LCD'ye yazdırılmıştır.

Ayrıca `if (input(pin_a0))` komutu ile RA0 butonuna basılıp basılmadığı kontrol edilmiş ve eğer bu butona basılmışsa `write_program_eeprom(adres, 0x0000);` komutu ile o an LCD'de gösterilen program adresinden bir sonraki adresten

başlanarak butona basıldığı sürece kod hafızası adreslerine 0x0000 bilgisi yazdırılmaktadır. Dolayısıyla program kodları silinmektedir. Bu işlemden sonra PIC'i resetlediğiniz de bir daha düzgün çalışmadığını görürsünüz. Çünkü program kodlarını silmiş olduk. PIC'in eski haline gelmesi için programı bir daha yüklemelisiniz. Şekil-15.4'de PIC program hafızası kodları ve uygulamanın LCD ekran sonucu görülmektedir. Görüldüğü gibi program hafızasındaki kod aynen LCD'de görülmektedir.



Şekil-15.4. *PIC program hafızası kodları ve uygulamanın LCD görüntüsü.*

16. Bölüm

ANALOG KARŞILAŞTIRICI VE VOLTAJ REFERANS MODÜLLERİ İŞLEMLERİ

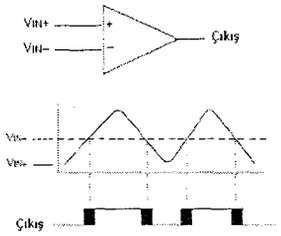
- Analog Karşılaştırıcı (Comparator) Modülü
- Karşılaştırma Kesmesi (#INT_COMP)
- SETUP_COMPARATOR() Fonksiyonu
- Karşılaştırıcı Voltaj Referans Modülü
- SETUP_VREF() Fonksiyonu
- Karşılaştırıcı ve Voltaj Referans Modülü Uygulaması

Bu bölümde bazı PIC denetleyici modellerinde bulunan karşılaştırıcı (comparator) ve voltaj referans modülleri incelenmiştir. Modüller tanıtıldıktan sonra da CCS C derleyicisinde bu modüllerin kontrolü için var olan fonksiyonlar tanıtılmıştır. En son olarak da bir uygulama ile teorik bilgiler pratiğe dökülmüştür. Uygulamada PIC16F877 yerine PIC16F877A kullanılmıştır. Çünkü PIC16F877 modelinde karşılaştırıcı ve referans modülleri bulunmamaktadır.

16.1. KARŞILAŞTIRICI (COMPARATOR) MODÜLÜ

Karşılaştırıcı birimi **iki adet** analog karşılaştırıcı içerir. Karşılaştırıcı modülü girişleri için RA0, RA1, RA2 ve RA3 pin'leri kullanılır. Bu pin'ler kullanılırken giriş olarak yönlendirilmelidir. Karşılaştırıcı modülü çıkışları olarak da RA4 ve RA5 pin'leri kullanılabilir. **RA4 pin'i açık kolektör yapısında olduğundan bu uç çıkış clarak kullanıldığında pull-up bağlantı yapılması gerekliliği hatırlanmalıdır.** Yine denetleyici içinde bulunan voltaj referans modülü de karşılaştırıcı modülü için giriş olarak kullanılabilir. Komparatör modülü ayarlarının yapıldığı kaydedici **CMCON** kaydedicisidir. Analog karşılaştırma işlemi Şekil-16.1'de görüldüğü gibi giriş olarak uygulanan iki analog sinyalden birinin diğerine göre gerilim seviyesinin karşılaştırılması suretiyle yapılır. Şekilde V_{IN-} referans voltajı

kabul edilmiştir. Çıkış dalga şeklinden görüldüğü gibi V_{IN+} sinyali gerilim değeri V_{IN-} sinyali gerilim değerinden küçük olduğu zaman çıkış dijital olarak lojik-0'dır. V_{IN+} sinyali gerilim değeri V_{IN-} sinyali gerilim değerinden büyük olduğu zaman çıkış dijital olarak lojik-1'dir. **Karşılaştırma modülü girişlerine uygulanan sinyallerin gerilim seviyesi VSS ile VDD gerilimleri (0V-5V) arasında olmalıdır.**



Şekil-16.1. *Karşılaştırma işlemi ve çıkış dalga şekli.*

CMCON KAYDEDİCİSİ

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

Şekil-16.2. *CMCON kaydedicisi*

Bit 7 **C2OUT** = Karşılaştırıcı–2 Çıkış Bit'i

C2INV=0 ise	C2INV=1 ise
1: C2 V_{IN+}>C2 V_{IN-}	1: C2 V_{IN+}<C2 V_{IN-}
0: C2 V_{IN+}<C2 V_{IN-}	0: C2 V_{IN+}>C2 V_{IN-}

Bit 6 **C1OUT** = Karşılaştırıcı–1 Çıkış Bit'i

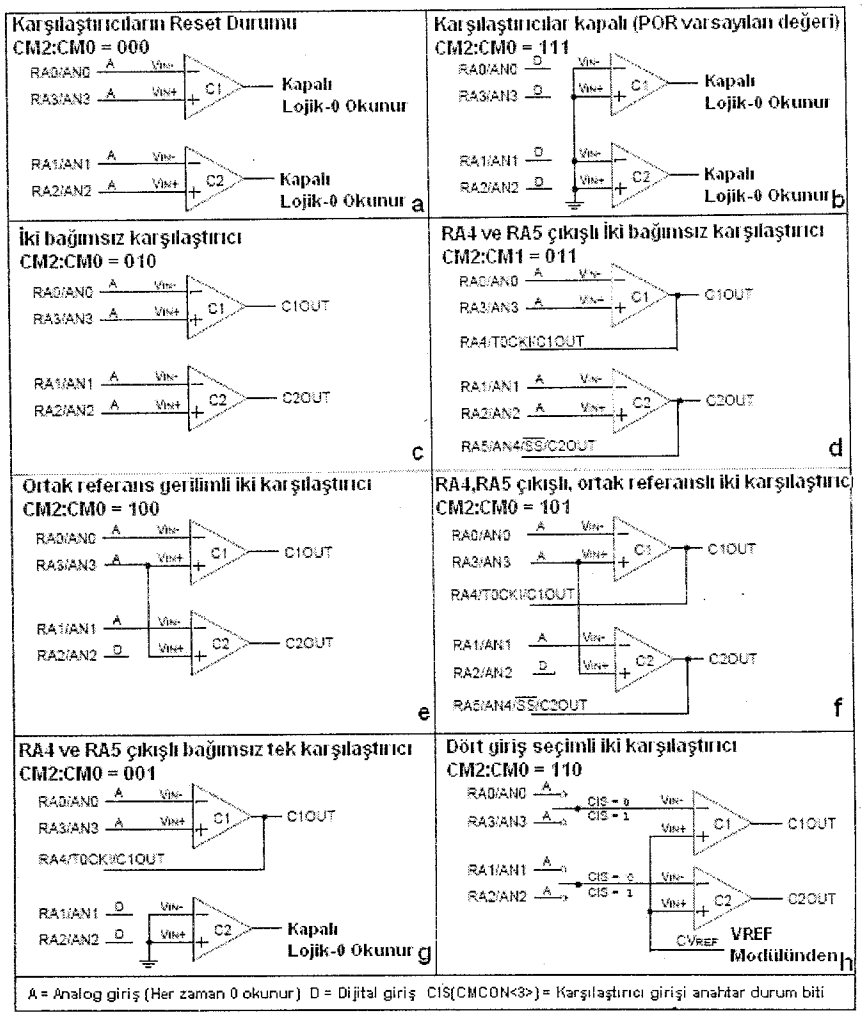
C1INV=0 ise	C1INV=1 ise
1: C1 V_{IN+}>C1 V_{IN-}	1: C1 V_{IN+}<C1 V_{IN-}
0: C1 V_{IN+}<C1 V_{IN-}	0: C1 V_{IN+}>C1 V_{IN-}

Bit 5 **C2INV** = Karşılaştırıcı–2 Çıkışını Tersleme Bit'i
1: Karşılaştırıcı–2 (C2) çıkışı terslenir.
0: Karşılaştırıcı–2 (C2) çıkışı terslenmez.

Bit 4 **C1INV** = Karşılaştırıcı–1 Çıkışını Tersleme Bit'i
1: Karşılaştırıcı–1 (C1) çıkışı terslenir.
0: Karşılaştırıcı–1 (C1) çıkışı terslenmez.

Bit 3 **CIS** = Karşılaştırıcı girişi anahtar durum bit'i (Sadece CM2:CM0 bit'leri 110 ise bu bit'in kullanılmasına ihtiyaç vardır).
CM2:CM0 = 110 ise
1: C1 V_{IN-} girişi RA3/AN3 pin'ine bağlanır
C2 V_{IN-} girişi RA2/AN2 pin'ine bağlanır
0: C1 V_{IN-} girişi RA0/AN0 pin"ine bağlanır
C2 V_{IN-} girişi RA1/AN1 pin'ine bağlanır

Bit 2,1,0 **CM2:CM0** = Karşılaştırıcı mod seçme bit'leri.
Şekil-16.3'de karşılaştırıcı modları ve bit değerleri verilmiştir.



Şekil-16.3. *CM2:CM1:CM0 bit'lerinin durumuna göre seçilen karşılaştırıcı modları.*

16.1.1. Karşılaştırıcı Kesmesi

Karşılaştırıcı modüllerinden herhangi birinin çıkış durumu değiştiğinde (lojik-0'dan lojik-1'e veya lojik-1'den lojik-0'a) karşılaştırıcı kesmesi meydana gelir. Bu nedenle kesmeyi aktif etmeden önce ilk çıkış durumları (CMCON kaydedicisinin 7. ve 6. bit'leri) kontrol edilmelidir. **PIC'in donanım yapısından dolayı Şekil-16.3'de verilen karşılaştırıcı mod seçimi yapılırken karşılaştırma kesmesi meydana gelmektedir. Bu nedenle mod seçim işlemi sırasında karşılaştırıcı kesmesi kapatılmalıdır. Aksi takdirde yanlış kesme sinyali oluşmuş olur.**

16.2. SETUP_COMPARATOR() FONKSİYONU

Bu fonksiyon, PIC denetleyicilerinde karşılaştırıcı modülü ayarlarını belirleyen fonksiyondur. Fonksiyondaki *"mod"* kısmına karşılaştırıcıların giriş-çıkış bağlantı sabitleri yazılır. Bu sabitler aşağıda verilmiştir. CMCON kaydedicisindeki Bit7 C2OUT ve Bit6 C1OUT karşılaştırıcı çıkış durumlarını bildiren bit'ler CCS C'de de **C1OUT** ve **C2OUT** değişkenleri isminde tanımlanmıştır. Denetleyicinin başlık dosyasında bu fonksiyonla ilgili kullanılabilecek sabitler görülebilir (C:\Program Files\PICC\Devices\ klasörü içinde. Örneğin 16F877A.h dosyası).

setup_comparator(*mod*);

Mod sabitleri;

NC_NC_NC_NC = Karşılaştırıcı birimi kapalı. Tüm girişler dijital *(Şekil-16.3-b)*
A0_A3_A1_A2 = Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3
Karşılaştırıcı–2 girişleri RA1/AN1 ve RA2/AN2 *(Şekil-16.3-c)*
A0_A3_A1_A2_OUT_ON_A4_A5 = Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3, karşılaştırıcı–1 çıkışı RA4. Karşılaştırıcı–2 girişleri RA1/AN1 ve RA2/AN2, karşılaştırıcı–2 çıkışı RA5/AN5. *(Şekil-16.3-d)*
A0_A3_A1_A3 = Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3, Karşılaştırıcı–2 girişleri RA1/AN1 ve RA3/AN3. *(Şekil-16.3-e)*
A0_A3_A1_A3_OUT_ON_A4_A5 = Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3, karşılaştırıcı–1 çıkışı RA4. Karşılaştırıcı–2 girişleri RA1/AN1 ve RA3/AN3, karşılaştırıcı–2 çıkışı RA5/AN5. *(Şekil-16.3-f)*
A0_A3_NC_NC_OUT_ON_A4 = Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3, karşılaştırıcı–1 çıkışı RA4 Karşılaştırıcı–2 bağlı değil. *(Şekil-16.3-g)*
A0_VR_A1_VR = Karşılaştırıcı–1 girişleri RA0/AN0 ve voltaj referans biriminden gelen CV_{REF} ucu. Karşılaştırıcı–2 girişleri RA1/AN1 ve voltaj referans biriminden gelen CV_{REF}ucu. *(Şekil-16.3-h, CIS bit'i 0 durumu)*
A3_VR_A2_VR = Karşılaştırıcı–1 girişleri RA3/AN3 ve voltaj referans biriminden gelen CV_{REF}(RA2/AN2) ucu. Karşılaştırıcı–2 girişleri RA2/AN2 ve voltaj referans biriminden gelen CV_{REF} ucu. *(Şekil-16.3-h, CIS bit'i 1 durumu)*

Örnek: setup_comparator(A0_A3_A1_A3);

16.3. KARŞILAŞTIRICI VOLTAJ REFERANS MODÜLÜ

Karşılaştırıcı voltaj referans modülü (Comparator Voltage Reference Module) 16 değişik kademede sabit referans voltajı sağlayan dâhili bir modüldür. Karşılaştırıcı voltaj referans modülü, karşılaştırıcı modülü birimi CM2:CM0 = 110, Şekil-16.3-h modunda seçildiğinde karşılaştırıcı girişlerine sabit voltaj referans kaynağı olarak kullanılır. Şekil-16.5'de karşılaştırıcı voltaj referans modülünün blok diyagramı görülmektedir. Karşılaştırıcı voltaj referans modülü **CVRCON** kaydedicisi ile kontrol edilir. Voltaj referans birimi güç kaynağı olarak (**CV_{RSRC}** olarak da tanımlanır) direkt

olarak PIC'in pozitif beslemesi V_{DD} ucunu kullanır. İstenirse karşılaştırıcı voltaj referans modülü çıkışı RA2/AN2/V_{REF-}/CV_{REF} pin'ine bağlanabilir. Bu sayede çıkış referans voltajı başka amaçlar için kullanılabilir. Özellikle bazı uygulamalarda mutlaka referans voltajına gereksinim duyulmaktadır. Bu işler için normalde voltaj referans entegreleri kullanılmaktadır. Bu uç sayesinde uygulamalar da ekstra bir entegre kullanmaktan kurtulmuş olunur.

CVRCON KAYDEDİCİSİ

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
bit 7							bit 0

Şekil-16.4. *CVRCON kaydedicisi.*

Bit7 **CVREN** = Karşılaştırıcı voltaj referans (CV_{REF}) modülü açma-kapama bit'i.
1: CV_{REF} modülü açık.
0: CV_{REF} modülü kapalı.

Bit 6 **CVROE** = CV_{REF} modülü V_{REF} çıkışı izin bit'i.
1: CV_{REF} modülü çıkış gerilimi RA2 pin'inde görülür.
0: CV_{REF} modülü çıkış gerilimi RA2 pin'inde görülmez.

Bit 5 **CVRR** = CV_{REF} modülü V_{REF} aralığı seçme bit'i.
1: 0'dan 0.75 CV_{RSRC}, CV_{RSRC}/24 adım ile
0: 0.25 CV_{RSRC}'den 0.75 CV_{RSRC}, CV_{RSRC}/32 adım ile

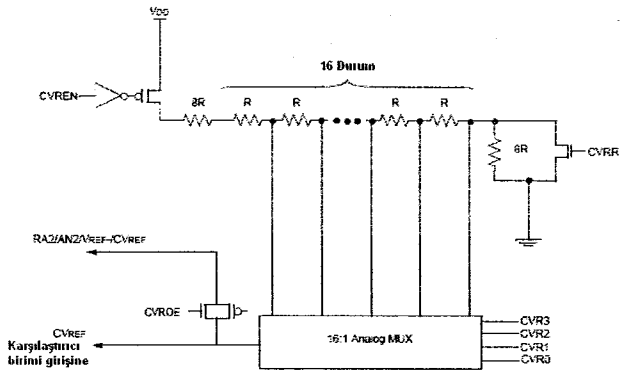
Bit 4 = Kullanılmayan bit. Okunduğunda "0" değeri alınır.

Bit 3:0 **CVR3:CVR0** = Karşılaştırıcı çıkış gerilimi olan V_{REF} gerilimi belirleme bit'leri.
Bu bit'lerin aldığı değer 0≤VR3:VR0≤15 arasında olmalıdır.

CVRR "1" ise
CV_{REF}=(<VR3:VR0>/24)xV_{DD}

CVRR "0" ise
V_{REF}=(V_{DD}/4)+(<VR3:VR0>/32)xV_{DD}

(V_{DD}, CV_{RSRC} olarak da ifade edilebilir.)



Şekil-16.5. *Karşılaştırıcı voltaj referans modülü blok diyagramı.*

16.4. SETUP_VREF() FONKSİYONU

Karşılaştırıcı voltaj referans modülünü kontrol etmeye yarayan fonksiyondur. Fonksiyondaki *"mod"* kısmına CVRCON kaydedicisinin 5. bit'inde olması istenen duruma göre sabit yazılır. Fonksiyondaki *"değer"* kısmına ise, V_{REF} çıkış voltajı hesaplamasında kullanılacak olan CVRCON kaydedicisinin Bit3:0 ile belirtilmesi istenen değer yazılır. 0 ile 15 arasında tam sayı tipinde bir değer olmalıdır. Fonksiyondaki iki sabitin arasına mutlaka | işareti konmalıdır. Denetleyicinin başlık dosyasında bu fonksiyonla ilgili kullanılabilecek sabitler görülebilir (C:\Program Files\PICC\Devices\ klasörü içinde. Örneğin 16F877.h dosyası).

setup_vref (*mod* | *değer*);

Mod sabitleri:

FALSE = Karşılaştırıcı referans birimi kapalı.
VREF_LOW = Bu mod seçilirse V_{REF} hesaplamasında CV_{REF}=(<VR3:VR0>/24)xV_{DD} formülü geçerli olur.
VREF_HIGH = Bu mod seçilirse V_{REF} hesaplamasında CV_{REF}=(V_{DD}/4)+(<VR3:VR0>/32)xV_{DD} formülü geçerli olur.
VREF_A2 = Bu mod seçilirse ayarlanan CV_{REF} gerilimi RA2 ucuna verilir.

```
setup_vref(VREF_LOW | 7);
```

Yukarıdaki komutla voltaj referans modülü çıkışı olan CV_{REF} aşağıdaki gibi bulunur.

CVREF=(<VR3:VR0>/24)xVDD=(7/24)x5=0,2917x5 1,46Volt olarak bulunur.

```
setup_vref(VREF_A2 | VREF_HIGH | 14);
```

Yukarıdaki komutla voltaj referans modülü çıkışı olan CV_{REF} aşağıdaki gibi bulunur ve bu değer RA2 ucunda görülür.

CVREF=(VDD/4)+(<VR3:VR0>/32)xVDD=(5/4)+(14/32)x5=1,25+(0,4375x5)
=1,25+2,1875 3,44Volt olarak bulunur.

16.5. KARŞILAŞTIRICI VE VOLTAJ REFERANS MODÜLÜ UYGULAMASI

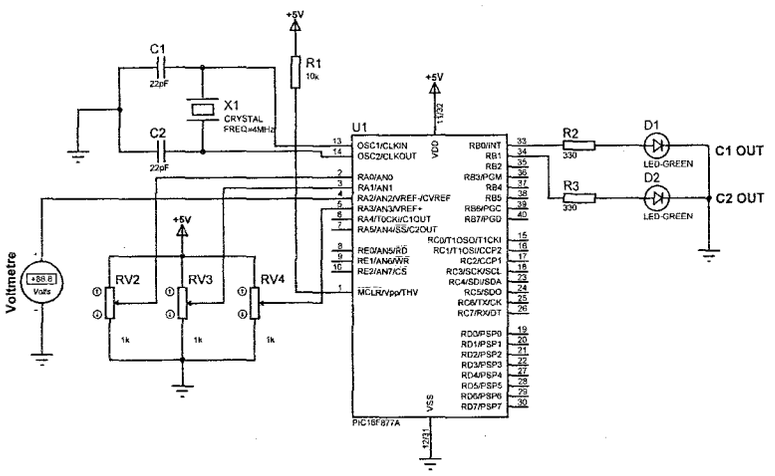
Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada hem karşılaştırıcı hem de voltaj referans modüllerinin kullanımı incelenmiştir. **PIC16F877 denetleyicisinde karşılaştırıcı ve voltaj referans modülleri olmadığından uygulamada PIC16F877A denetleyicisi kullanılmıştır.** RA0, RA1 ve RA3 girişlerine potansiyometre bağlanmıştır. Karşılaştırıcı modülü ortak referanslı (RA3) iki girişli (RA0 ve RA1) moda ayarlanmıştır. Böylece RA3 ucuna gelen gerilim RA0 ve RA1 uçları için referans

gerilim olmuştur. RB0 ve RB1 çıkışlarına LED bağlanmıştır. Böylece karşılaştırma sonuçları LED'lerde görünmektedir. **CCS C derleyicisi karşılaştırıcı modülünü şu şekilde kurmaktadır. Giriş gerilimi referans geriliminden daha büyük ise çıkış (C1OUT veya C2OUT) lojik-0, giriş gerilimi referans geriliminden daha küçük ise çıkış (C1OUT veya C2OUT) lojik-1 olacak şekilde programlanmaktadır.** Uygulamada ayrıca voltaj referans modülü de kullanılmıştır. Bu modül RA2 çıkışına yaklaşık 3,6V verecek şekilde ayarlanmıştır.

- Devrenizi PIC PROG/DEKA PIC16F87x deneme kartınızda denemek için, deneme kartının sol tarafında bulunan PortA konnektöründe RA0, RA1 ve RA3 pin'lerine zil teli ile istediğiniz gerilimleri uygulayınız. Bu uçlara deneme kartınızın VDD ve GND uçlarından aldığınız gerilimleri bağlayarak çalışmanızı deneyebileceğiniz gibi, bir board'a 3 adet pot yerleştirerek buradan aldığınız değişik gerilim değerleri ile de devrenizi deneyebilirsiniz.

- Deneme kartında RA2 ucunda LED ve direnç bağlantısı olduğundan, RA2 ucundan okunan gerilim hesaplanan gerilimden düşük olabilir.



Şekil-16.6. *Karşılaştırıcı ve Voltaj refereans modülü uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877A ile Karşılaştırıcı Modülü Uygulaması
PIC PROG/DEKA      : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877A.h>      // Kullanılacak denetleyicinin başlık
                          // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use fast_io(a) //Port yönlendirme komutları A port'u için geçerli
```

```
#use fast_io(b) //Port yönlendirme komutları B port'u için geçerli

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   set_tris_a(0x0B);    // RA0,RA1,RA3 pin'i giriş,RA2 çıkış
   set_tris_b(0x00);    // B port'u komple çıkış
   output_b(0x00);      // İlk anda B port'u çıkışını sıfırla

   setup_comparator(A0_A3_A1_A3);//Karşılaştırıcı ayarları yapılıyor
   setup_vref (VREF_A2 | VREF_HIGH | 15); //Voltaj Referans Modülü
                                          // ayarları yapılıyor
   while (1)  // Sonsuz döngü
   {
      if (C1OUT==1)  // Karşılaştırıcı-1 çıkışı "1" ise
         output_high(pin_b0);  // RB0 çıkışı lojik-1
      else
         output_low(pin_b0);   // RB0 çıkışı lojik-0

      if (C2OUT==1)  // Karşılaştırıcı-2 çıkışı "1" ise
         output_high(pin_b1); // RB1 çıkışı lojik-1
      else
         output_low(pin_b1);  // RB1 çıkışı lojik-0
   }
}
//*****************************************************************
```

Programda `setup_comparator(A0_A3_A1_A3);` komutu ile karşılaştırıcı modülü, Karşılaştırıcı–1 girişleri RA0/AN0 ve RA3/AN3, Karşılaştırıcı–2 girişleri RA1/AN1 ve RA3/AN3 (Şekil-16.3-e) olacak şekilde ayarlanmıştır. `setup_vref (VREF_A2 | VREF_HIGH |15);` komutu ile de voltaj referans modülü RA2 pin'ine ayarlanan gerilim çıkışını verecek şekilde kurulmuştur. Voltaj referans modülünde ayarlanan gerilim yaklaşık 3,6 volttur.

CVREF=(VDD/4)+(<VR3:VR0>/32)xVDD=(5/4)+(15/32)x5=1,25+(0,46875x5)
=1,25+2,34375 3,59Volt olarak bulunur.

Sonsuz döngüde 1.karşılaştırıcı çıkışı C1OUT ve 2.karşılaştırıcı çıkışı C2OUT çıkışlarının durumlarına göre RB0 ve RB1 uçlarına bağlı LED'ler yakılıp söndürülmektedir. Örneğin bu uygulamada RA0 girişindeki gerilim referans gerilimi olan RA3 ucundaki gerilimden küçük ise C1OUT lojik-1, RA0 girişindeki gerilim referans gerilimi olan RA3 ucundaki gerilimden büyük ise C1OUT lojik-0 olacaktır.

17. Bölüm

PARALEL SLAVE PORT (PSP) MODÜLÜ İŞLEMLERİ

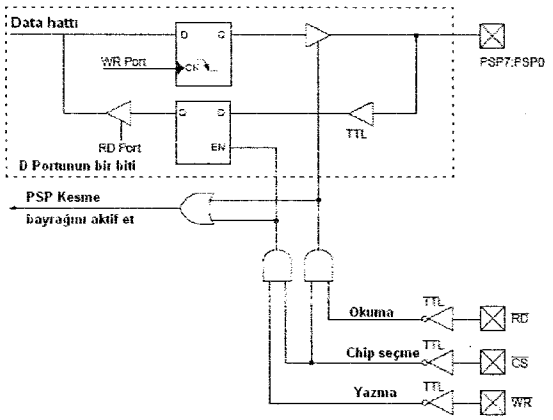
- Paralel Slave Port (PSP) Modülü
- PSP Kesmesi (#INT_PSP)
- SETUP_PSP() Fonksiyonu
- PSP_OUTPUT_FULL() Fonksiyonu
- PSP_INPUT_FULL() Fonksiyonu
- PSP_OVERFLOW() Fonksiyonu
- Paralel Slave Port Modülü Uygulaması

Bu bölümde bazı PIC denetleyici modellerinde bulunan Paralel Slave Port modülü incelenmiştir. PSP modülü genel olarak tanıtıldıktan sonra CCS C derleyicisinde PSP modülü ayarları ve kullanımı ile ilgili fonksiyonlar tanıtılmıştır. En son olarak da bir PSP modülü uygulaması gerçekleştirilmiştir.

17.1. PARALEL SLAVE PORT (PSP) MODÜLÜ

PSP modülü denetleyicinin başka bir cihaz ile direkt olarak ile 8 bit'lik bir data yolundan iletişim kurmasına izin verir. Bu özellik sayesinde denetleyici başka bir 8 bit'lik veri yoluna sahip denetleyici ile veya başka bir eleman ile kolayca iletişim sağlayabilir. PSP modülü D port'unu data yolu (RD0/PSP0, RD1/PSP1...RD7/PSP7), E port'u pin'lerini ise kontrol uçları olarak (RE0/$\overline{RD}$, RE1/$\overline{WR}$, RE2/$\overline{CS}$) kullanır. PSP modülü pin'leri (D ve E port'u) aynı zamanda başka amaçlar için de kullanıldığından, bu port'lar hangi amaçla kullanılacaksa ona göre yönlendirilmelidirler. Örneğin E port'u aynı zamanda analog giriş için kullanıldığından bu port'un dijital giriş/çıkış için kurulması gerekir. PSP modülü ile ilgili ayarlar **TRISE** kaydedici ile yapılır.

PSP modülü çalışması slave mod bir çalışmadır. **Kontrol sinyalleri denetleyici tarafından üretilmez.** İletişim kurulan diğer eleman kontrol sinyallerini üretir. Denetleyici sadece kontrol sinyallerini okuyarak, durumlarına göre işlem yapar.



Şekil-17.1. *D ve E port'u PSP Modülü blok diyagramı.*

TRISE KAYDEDİCİSİ

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0
bit 7							bit 0

Şekil-17.2. *TRISE kaydedicisi.*

Bit7 **IBF** = Giriş tamponu (buffer) dolu durum bit'i
1: Data alınmış ve denetleyicinin okumasını bekliyor.
0: Daha bilgi alınmamış.

Bit 6 **OBF** = Çıkış tamponu (buffer) dolu durum bit'i
1: Çıkış tamponu hala bir önceki data'yı tutuyor.
0: Çıkış tamponu okuma yapıyor.

Bit 5 **IBOV** = Giriş tamponu taşma algılama durum bit'i
1: Bir önceki giriş data'sı okunmadan yazma işlemi meydana gelmiş.
0: Taşma meydana gelmemiş.

Bit 4 **PSPMODE** = Paralel Slave Port Modülü seçme bit'i
1: PSP modülü açık.
0: PSP modülü kapalı.

Bit 3 Kullanılmayan bit. Okunduğunda "0" değeri alınır.

Bit 2 **TRISE2** = RE2 pin'i yönlendirme kontrol bit'i
1: RE2 giriş.
0: RE2 çıkış.

Bit 2 **TRISE1** = RE1 pin'i yönlendirme kontrol bit'i
1: RE1 giriş.
0: RE1 çıkış.

Bit 2 **TRISE0** = RE0 pin'i yönlendirme kontrol bit'i
1: RE0 giriş.
0: RE20 çıkış.

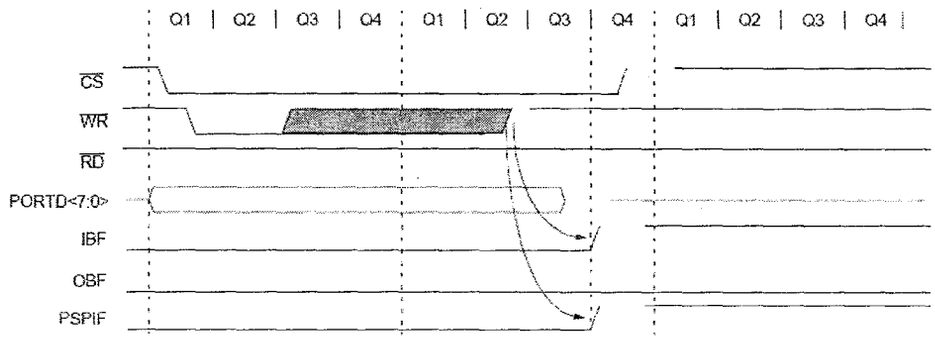
PSP modülünde D port'u üzerinden hem veri yazımı hem de okuma işlemi yapılır. Okuma veya yazma işlemlerini E port'unun bit'leri belirler. RE0/$\overline{RD}$ ve RE1/$\overline{WR}$ uçlarının durumu belirler. Tablo-17.1'de bu işlemler özetlenmiştir.

Pin İsmi	Açıklama
RE0/ $\overline{RD}$	PSP modunda okuma işlemini kontrol eder. 1: Okuma yok. 0: Port D kaydedicisini okur. (Eğer chip seçilmişse)
RE1/ $\overline{WR}$	PSP modunda yazma işlemini kontrol eder. 1: Yazma yok. 0: Port D kaydedicisine bilgi yazar. (Eğer chip seçilmişse)
RE2/ $\overline{CS}$	PSP modunda chip seçme işlemini yapar. 1: Chip seçilmemiş. 0: Chip seçilmiş.

Tablo-17.1. *PSP modunda E port'u bit'lerinin durumları ve açıklamaları.*

17.1.1. PSP Modülü Yazma İşlemi

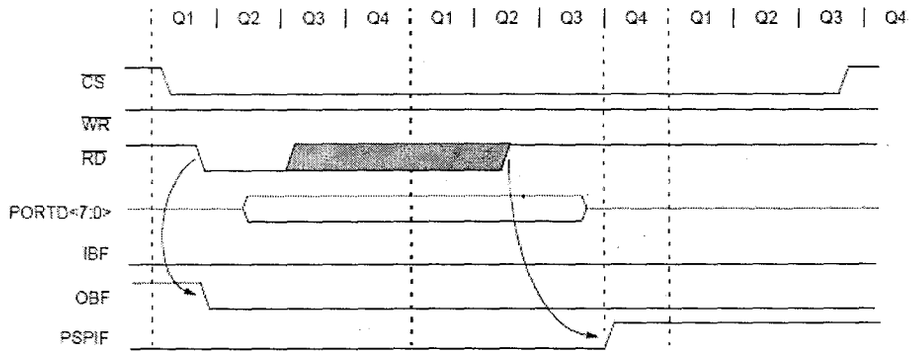
PSP modülüne dışarıdan herhangi bir cihazdan yazma işleminin yapılabilmesi için, dış cihazdan veri PSP data port'u olan D port'una gönderilir. RE2/$\overline{CS}$ ve RE1/$\overline{WR}$ ilk başta lojik-0 iken yazma işlemi meydana gelir. RE2/$\overline{CS}$ veya RE1/$\overline{WR}$ uçlarından biri lojik-1 olunca (yükselen kenar tetikleme) yazma işlemi tamamlanır ve TRISE kaydedicisindeki giriş tamponu dolu (Input Buffer Full - IBF) bit'i "1" olur. Eğer PSP kesmesi aktif ise aynı zamanda bu işlem sonucunda PSP kesmesi (PSPIF) meydana gelir (Bilindiği gibi kesme bayrakları meydana geldikten sonra silinmesi gerekir. CCS C bu işlemi otomatik olarak yapmaktadır). PSP hattındaki veri okununca IBF bit'i "0" olur. Eğer tampon'daki veri okunmamışsa IBF bit'i "0" olmaz ve bu anda yeni bir yazma işlemi meydana gelirse TRISE kaydedicisinin 5.bit'i olan IBOV giriş tamponu taşma bit'i "1" olur. Bu işlemlerin dalga şekilleri ile gösterimi Şekil-17-3'de görülmektedir.



Şekil-17.3. *PSP yazma işlemi dalga şekilleri.*

17.1.2. PSP Modülü Okuma İşlemi

PSP modülünde okuma işleminin yapılabilmesi için ilk başta RE2/$\overline{CS}$ ve RE0/$\overline{RD}$ bit'lerinin ikisinin de lojik-0 olması gerekir. Bu durumda TRISE kaydedicisinin 6. bit'i olan OBF (çıkış tampon dolu) bit'i hemen okuma işleminin olduğunu belirtmek için "0" olur. RE2/$\overline{CS}$ veya RE0/$\overline{RD}$ uçlarından biri lojik-1 olunca (yükselen kenar tetikleme) okuma işlemi tamamlanır ve kesme aktifse PSP kesmesi (PSPIF) meydana gelir. Bilgi kullanıcı (program) tarafından D port'una yazılana kadar OBF bit'i "0" durumunu korur. Bu işlemlerin dalga şekilleri ile gösterimi Şekil-17.4'de görülmektedir.



Şekil-17.4. *PSP okuma işlemi dalga şekilleri.*

17.1.3. Paralel Slave Port Kesmesi (#INT_PSP)

PSP kesmesi aktif edilmiş ise, her okuma ve yazma işlemi bittiğinde kesme meydana gelir. PSP kesmesinin kodu #INT_PSP'dir. Diğer kesme işlemleri ile aynı mantıkla kontrol edilir.

17.2. SETUP_PSP() FONKSİYONU

PSP modülünün açılıp kapatılmasını sağlayan fonksiyondur. Fonksiyondaki *"mod"* kısmına aşağıda verilen sabitlerden biri yazılır. Denetleyicinin başlık dosyasında bu fonksiyonla ilgili kullanılabilecek sabitler görülebilir (C:\Program Files\PICC\Devices\ klasörü içinde. Örneğin 16F877.h dosyası).

setup_psp(***mod*** **);**

MOD Sabitleri:

PSP_ENABLED = PSP modülü açık.
PSP_DISABLED = PSP modülü kapalı.

```
setup_psp(psp_enabled); // PSP modülü açık
setup_psp(psp_disabled); // PSP modülü kapalı
```

PSP modülü işlemlerinde D port'unundaki veriyi okumak için input_d(), D port'una veri göndermek için de output_d komutları kullanılabilir. Fakat CCS C'de PSP modülü kullanımı için D port'unu ifade eden **PSP_DATA** değişkeni tanımlanmıştır.

17.3. PSP_INPUT_FULL(), PSP_OUTPUT_FULL(), PSP_OVERFLOW() FONKSİYONLARI

Bu fonksiyonlar PSP iletişimi sırasında giriş/çıkış tamponlarını kontrol etmek ve herhangi bir PSP tampon taşma durumunu kontrol etmeye yarayan fonksiyonlardır. Denetleyicinin başlık dosyasında bu fonksiyonla ilgili kullanılabilecek sabitler görülebilir (C:\Program Files\PICC\Devices\ klasörü içinde. Örneğin 16F877.h dosyası).

PSP_INPUT_FULL() = Giriş tamponu doluysa (bilginin denetleyici tarafından okunması bekleniyorsa) bu fonksiyon "1" bilgisi ile geri döner. Aksi halde fonksiyon "0" bilgisi ile geri döner.
PSP_OUTPUT_FULL() = Çıkış tamponu doluysa (bilginin dış eleman tarafından okunması bekleniyorsa) bu fonksiyon "1" bilgisi ile geri döner. Aksi halde fonksiyon "0" bilgisi ile geri döner.
PSP_OVERFLOW() = Giriş tamponu taşması meydana gelmişse bu fonksiyon "1" bilgisi ile geri döner. Aksi halde fonksiyon "0" bilgisi ile geri döner.

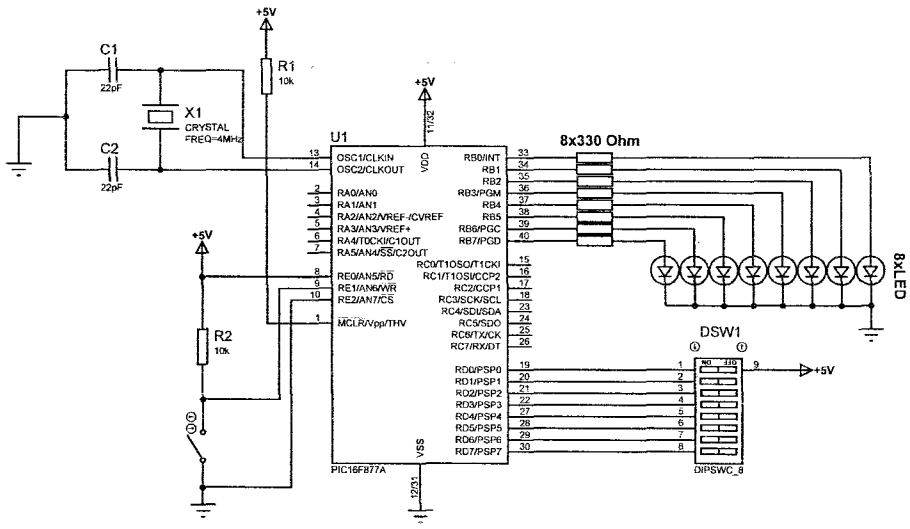
durum=psp_overflow();//Taşma varsa "*durum*"değişkeni "1" yoksa "0" olur.

17.4. PARALEL SLAVE PORT MODÜLÜ UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada PSP modülünün kullanılması incelenmiştir. Uygulamada D port'una 8'li dip switch, B port'una ise 8 adet LED bağlanmıştır. Dip switch'in ana ucu +5V'a bağlanmıştır. Dip switch ile D port'u pin'lerine "1" veya "0" bilgisi verilmektedir. RE0/$\overline{RD}$ ucu +5V'a bağlanmıştır, chip seçme ucu RE2/$\overline{CS}$ ise şaseye bağlanmıştır. Böylece PSP modülü yazma moduna ayarlanmıştır. Yazma RE1/$\overline{WR}$ ucuna bağlanan anahtar bağlantısı ile bu uca "1" ve "0" bilgisi verilebilmektedir. İlk başta dip switch ile gönderilmek istenen veri ayarlanır. Daha sonra RE1/$\overline{WR}$ ucunun bağlı bulunduğu anahtar kapatılır. Böylece yazma ucu ve chip seçme ucunun ikisi de lojik-0 olmuş olur. Daha sonra anahtar açıldığında yazma ucu lojik-0'dan lojik-1 seviyesine (yükselen kenar) geldiğinden yazma işlemi gerçekleşir. Dip switch ile ayarlanan bilgiler B port'undaki LED'lerde görünür. Dip switch'in anahtarlarının konumu değiştirilerek bu şekilde istenen veri PSP modülü ile alınmış olunur.

Normalde dip switch yerine 8 bit ile veri yolu ile iletişim kuran bir eleman olmalıdır. PSP modülü özellikle PC paralel port iletişimi ile metin dosyaların denetleyiciye iletimi için rahatlıkla kullanılabilir. **CCS C'de PSP modülü kullanımı oldukça basittir. PSP modülünü kullanıma açmak çoğu PSP ayarlarının yapılması için yeterlidir.**



Şekil-17.5. *PSP Modülü uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877A ile Paralel Slave Port Modülü Uygulaması
PIC PROG/DEKA     : Port B jumper'ı LED konumunda olmalı
*****************************************************************/
#include <16f877A.h>      // Kullanılacak denetleyicinin başlık
                          // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
osilatör frekansı belirtiliyor.

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı
   setup_comparator(NC_NC_NC_NC);// Karşılaştırıcı birimi devre dışı
```

```
    setup_VREF(FALSE);            // Voltaj referans birimi devre dışı

    setup_psp(PSP_ENABLED);  // PSP modülü açık

    while (1)  // Sonsuz döngü
    {
     output_b(PSP_DATA);//PSP veri yolundaki bilgiyi B portuna gönder
    }
}
//*****************************************************************
```

Programda dikkat edilirse #use fast_io() komutu kullanılmamıştır. Dolayısıyla port yönlendirme komutu set_tris_x() da kullanılmamıştır. Önceden belirtildiği gibi programda #use fast_io() komutu kullanılmadığında derleyici varsayılan olarak #use standart_io() komutunun kullanıldığını varsayar. Böylece CCS C port'ların kullanımına göre, otomatik olarak port pin'lerini yönlendirir. Bizim programda set_tris_x() komutu ile hangi port'un hangi pin'lerinin giriş/çıkış olacağını bildirmemize gerek kalmaz.

`setup_psp(PSP_ENABLED);` komutu ile PSP modülü açılmıştır (aktif edilmiştir). Daha sonra sonsuz döngüde `output_b(PSP_DATA);` komutu ile D port'undaki bilgi PSP modülü kontrolünde B port'una aktarılmaktadır. CCS C'de `PSP_DATA` değişkeni D port'unu ifade eder. Kullanılan bu komut yerine `output_b(input_d());` komutu da kullanılabilirdi.

NOTLAR:

18. Bölüm

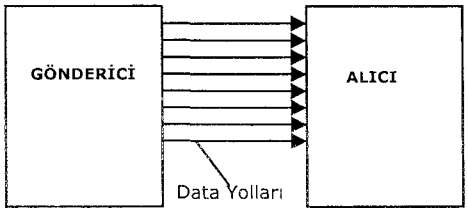
RS232 SERİ İLETİŞİM İŞLEMLERİ

- RS232 Seri İletişim
- MAX232 Entegresi
- RS232 Seri İletişim Kesmeleri (#INT_RDA, #INT_TBE)
- #USE RS232 () Fonksiyonu
- SET_UART_SPEED() Fonksiyonu
- RS232 Giriş/Çıkış Fonksiyonları
- RS232 Seri İletişim Uygulaması

Bu bölümde RS232 seri iletişim hakkında bilgi verilmiş ardından bu iletişim için gerekli donanım entegrelerinden biri olan MAX232 entegresi tanıtılmıştır. Konu ile ilgili genel bilgilerden sonra CCS C'de RS232 işlemlerinin nasıl yapıldığı anlatılmıştır. En son olarak da RS232 iletişimi ile ilgili bir uygulama gerçekleştirilmiştir.

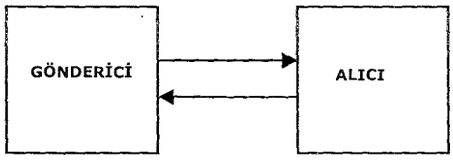
18.1. RS232 SERİ İLETİŞİM

Bilgisayarlar ile iletişim kurabilmek için bir takım yöntemler vardır. Seri haberleşme, paralel haberleşme gibi. Paralel iletişimde her bir bilgi için bir data hattı vardır. Yani 8 bit'lik bir veri gönderimi için 8 tane data hattına ihtiyaç vardır. Paralel haberleşmede tüm bit'ler aynı anda data yollarından gönderilir. Bu nedenle paralel iletişim seri iletişime göre daha hızlıdır. Fakat paralel iletişim ile uzun mesafelere veri iletmek hem uygun değildir, hem de pahalıdır.



Şekil-18.1. *Paralel İletişim.*

Bu sebeple seri iletişim daha yavaş olmasına rağmen, hem daha ucuz hem de uzun mesafelere iletim için uygundur. Seri haberleşmede bilgiler tek bir kanal üzerinden gönderilip, tek bir kanal üzerinden alınır.



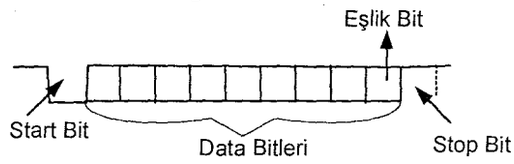
Şekil-18.2. *Seri İletişim.*

Seri haberleşmenin iki yöntemi vardır.

1- Senkron (eşzamanlı) 2- Asenkron (eşzamansız)

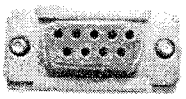
Senkron seri iletişimde gönderilen veri bit'leri ile alınan veri bit'leri arasında bir uyum olmalıdır. Yani senkron ifadesi, alıcı devre ile verici devrenin eş zamanlı çalışması anlamına gelmektedir. Bu alış-veriş arasındaki uyumu sağlamak için data kablosu yanında birde clock sinyali kablosu bulunur. Clock sinyalinin periyodu seri iletişimdeki her bir birimin iletim süresini belirtir. Böylece bilgi iletişimi için start (başlangıç) ve stop (bitiş) bit'lerine gerek kalmaz. Senkron seri iletişim asenkron seri iletişime göre daha hızlıdır. Ancak senkron iletişim karmaşık ve pahalı devreler içerir.

Asenkron seri iletişimde ise sadece veri hattı bulunur. Alıcı ve verici devreler veri iletişiminde eş zamanlı çalışmazlar. Senkron iletimdeki clock palsi yerine asenkron iletişimde *start bit'i, stop bit'i* ve *parity bit'i* vardır. **Start (başlangıç) bit'i,** bilginin gönderilmeye başlandığını alıcı tarafa bildirmek için kullanılır. Asenkron iletişimin ilk bit'i her zaman start bit'idir. **Stop (bitiş) bit'i,** gönderilen bilginin bittiğini alıcı tarafa belirtir. Stop bit'inden sonra gönderilen bilgi yeniden start bit'i ile başlar. **Parity (eşlik) bit'i,** alıcı tarafa gönderilen karakterin doğru olarak iletilip iletilmediğini kontrol etmek için kullanılan bit'tir. Asenkron iletişimde ODD, EVEN, MARK, SPACE, veya NONE parity bit'leri kullanılır. Parity bit'inin asenkron iletişimde kullanılması zorunlu değildir. **Data bit'ler,** gönderilecek olan bilgiyi oluşturan bit'lerdir. Gönderilecek bilgi 7 veya 8 bit olabilir. Asenkron veri iletişiminde start bit'i her zaman lojik-0, stop bit'i ise her zaman lojik-1 olmalıdır. Şekil-18.3'de asenkron veri formatı görülmektedir.



Şekil-18.3. *Asenkron iletişim veri formatı.*

Seri iletişim hızı, saniyede gönderilen bit sayısı olarak (**bps-bit per second**) tanımlanır. Bu orana seri iletişimin hızını belirtmek için kullanılan baud oranı (baud rate) denir. Seri iletişim baud oranları, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 olarak çeşitli değerlerdedir. Bilgisayarlarda Asenkron seri iletişim için 9 ve 25 uçlu RS-232 C port'ları vardır. Günümüzde daha çok 9 pin'li seri port kullanılmaktadır. Şekil-18.4'de 9 pin'li dişi ve erkek seri port'ları görülmektedir.



Dişi(Female)

Erkek (Male)

Şekil-18.4. *9 pin'li erkek ve dişi seri port'ları.*

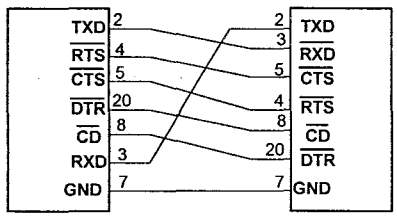
9 pin'li RS-232 port'unun pin'lerinin açıklamaları Tablo-18.1'de verilmiştir. Şekil-18.5'de ise şekilsel olarak pin uçları görülmektedir. 9 pin bulunmasına rağmen 3 uçla iki tane RS-232 birimi birbiri ile bilgi alış-verişi kurabilir. Bu pin'ler TD, RD, GND uçlarıdır. Sadece bu uçları kullanarak basit bir seri iletişim gerçekleştirilebilir.

Pin No	Sembol	Açıklama
1	CD	Veri Taşıyıcı Algılayıcı (Data Carrier Detect)
2	RD-RXD	Veri Alma (Receive Data)
3	TD-TXD	Veri Gönderme (Transmit Data)
4	DTR	Veri Terminal Hazır (Data Terminal Ready)
5	GND	Toprak (Signal Ground)
6	DSR	Veri Hazır (Data Set Ready)
7	RTS	Gönderme İsteği (Request To Send)
8	CTS	Göndermek İçin Sil (Clear To Send)
9	RI	Halka Gösterici (Ring Indicator)

Tablo-18.1. *9 pin'li RS-232 port'unun pin'lerinin açıklamaları.*

Şekil-18.5. *9 pin'li Dişi (DB9F) ve Erkek (DB9M) seri port pin numaraları.*

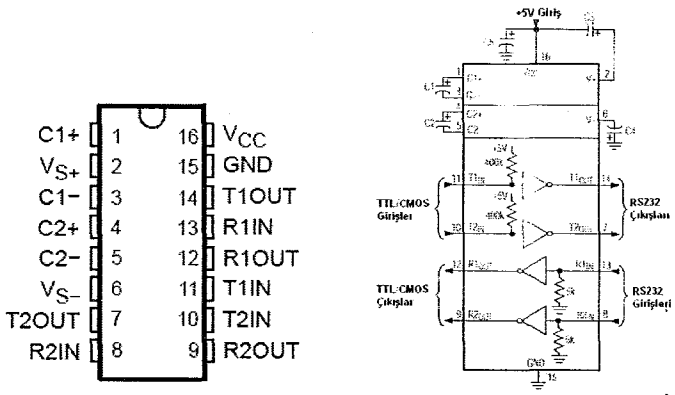
İki RS-232 alıcı-verici devre arasında DB9 port'unun pin bağlantıları Şekil-18.6'da verildiği gibi olmalıdır. Dikkat edilirse bu bağlantıda alma ve gönderme uçları çapraz bağlanır. **Gönderilen verilerin karşı taraftan doğru alınıp alınmadığının kontrolü yapılmak istenmiyorsa sadece TXD, RXD, GND uçlarının bağlantısıyla seri iletişim gerçekleştirileni bilir. Fakat bu seçim yüksek baud oranlı seri iletişimlerde hatalara meydan verebilir.**



Şekil-18.6. *İki alıcı-verici arasında seri iletişim pin bağlantıları.*

18.2. MAX232 ENTEGRESİ

RS-232 iletişim standardı TTL entegrelerinden daha önce belirlendiğinden gerilim seviyeleri TTL uyumlu değildir. Yani RS-232'de lojik-0, +3 ile 25 volt arasında, lojik-1 ise -3 ile 25 volt arasında tanımlanmaktadır. Fakat mikrodenetleyiciler TTL uyumludur. Lojik-1 yaklaşık 3-5 volt, lojik-0 ise yaklaşık 0-2,5 volt arasında tanımlanmaktadır. Dolayısı ile RS-232 mikrodenetleyici ile direkt olarak iletişim kuramaz. Bu nedenle iletişimi sağlamak amacı ile gerilim dönüştürücü devreler veya entegreler kullanmak gerekir. Bu problemin çözümü için en çok kullanılan entegre ise MAX232C gerilim dönüştürücü entegresidir. Şekil-18.7'de MAX232 entegresinin pin'leri ve genel bağlantı şekli görülmektedir. **Şekilde görülen kondansatörlerin tümü 1 µF'tır.** Entegrede 2 giriş ve 2 çıkış ucu vardır. RS232 port'una bilgi gönderme (TX) uçları 14 ve 7 (T1OUT, T2OUT) uçlarıdır. Bu uçlardan biri RS232 port'unun alma (RX) ucuna bağlanır. Entegrenin RS232 port'undan bilgi alıcı (RX) uçları,13 ve 8 ($R1_{IN}$, $R2_{IN}$) uçlarıdır. Bu uçlardan biri RS232 port'unun veri gönderme ucuna (TX) bağlanır. Mikrodenetleyiciden veya herhangi bir entegreden gelen bilgileri alan uç $T1_{IN}$ veya $T2_{IN}$, gönderen uç ise $R1_{OUT}$ veya $R2_{OUT}$ uçlarıdır. Kondansatörlerin kutupsuz seramik kondansatörler olması daha uygundur. Fakat kutuplu elektrolit kondansatörler kullanılacaksa polarite bağlantıları şekilde verildiği gibi olmalıdır.



Şekil-18.7. *MAX232 Entegtesi bacakları ve bağlantıları.*

18.3. RS232 SERİ İLETİŞİM KESMELERİ

#INT_RDA kesmesi seri denetleyicinin seri haberleşme veri alma ucuna (RX) bilgi geldiği zaman aktif olur. Kesmenin kullanımı önceden anlatılan kesme işlemleri ile aynıdır. **Fakat bu kesmede kesme fonksiyonunda, kesme pasif haline getirilmelidir. Aksi takdirde program hep kesmeye gidecektir.** Bazı denetleyicilerde INT_RDA0, INT_RDA1, INT_RDA2 kesmeleri de bulunmaktadır. Aynı zamanda gönderme tamponu boş olduğunda meydana gelen **#INT_TBE**(**T**ransmit **B**uffer **E**mpty) kesmesi de mevcuttur.

18.4. #USE RS232() FONKSİYONU

Bu komut derleyiciye hangi pin'lerin seri iletişim için kullanılacağını, seri iletişim hızının ne olacağını, iletişim için parity bit'inin ne olacağı gibi seri iletişim ile ilgili ayarları derleyiciye bildiren ön işlemci komutudur. Fonksiyondaki *"sabitler"* kısmına aşağıda verilen sabitlerden kullanılması gerekenler yazılır. Her sabit virgül ile birbirinden ayrılır. **PIC16F877'nin içinde RS232 iletişimi için UART donanım modülü vardır. Bu modülü kullanarak RS232 iletişimi belli ve sabit pin'lerle yapılır. Fakat CCS C derleyicisi bize istediğimiz pin'leri kullanarak RS232 iletişimi de sağlar.**

#use rs232 (*sabit, sabit, sabit,...*)

<u>#USE RS232 Sabitleri</u>

baud=x RS232 iletişim hızını belirten sabittir. (baud=9600 gibi)
xmit=pin RS232 veri gönderme ucunu belirten sabittir. (xmit=pin_A2 gibi)
rcv=pin RS232 veri alma ucunu belirten sabittir. (rcv=pin_A3 gibi)
parity=x RS232 iletişimi eşlik bit'i durumunu bildirir. X yerine N (eşlik bit'i yok), E (eşlik bit'i çift), O (eşlik bit'i tek). (parity=N gibi)
stop=x RS232 stop bit'inin kaç bit olduğunu belirtir. X yerine 1 veya 2 yazılır. Varsayılan stop bit'i 1'dir. (stop=1 gibi)
invert RS232 pin'lerinin polaritesini değiştirir. Yani xmit ile rcv pin'leri yer değiştirir.
UART1 xmit ve rcv pin'leri olarak denetleyici içinde bulunan birinci UART modülü pin'lerinin kullanılacağını belirtir.
UART2 xmit ve rcv pin'leri olarak denetleyici içinde bulunan ikinci UART modülü pin'lerinin kullanılacağını belirtir.
RESTART_WDT = getc() fonksiyonu veri okumayı beklerken WDT'ı sıfırlayarak, WDT'ın denetleyiciyi resetlemesini önler.

```
#use rs232 (baud=9600, xmit=PIN_A2, rcv=PIN_A3,stop=1,parity=N)
```

Bu komut ile iletişim hızının 9600 olduğunu, veri gönderme ucunun A port'unun 2.bit'i, veri alma ucunun A port'unun 3.bit'i olduğunu, stop bit'inin 1 bit olacağını, eşlik bit'inin olmayacağını derleyiciye bildirmiş oluyoruz.

18.5. SET_UART_SPEED() FONKSİYONU

Bu fonksiyon çalışma anında RS232 iletişim hızını (baud rate) değiştirme işini sağlayan fonksiyondur.

set_uart_speed(*baud hızı*);

```
set_uart_speed(9600);   // Seri iletişim baud hızı 9600
set_uart_speed(19200);  // Seri iletişim baud hızı 19200
```

18.6. RS232 GİRİŞ/ÇIKIŞ FONKSİYONLARI

RS232 iletişimi sırasında karakter alımı - gönderimi, string (ifade) alımı - gönderimi gibi işlemlerin yapılmasını sağlayan fonksiyonlardır. Bu fonksiyonların kullanılabilmesi için programın başında #use rs232 ön işlemci fonksiyonunun tanıtılması gerekmektedir. Temel C dilinde olan ve CCS C'de kullanılan Giriş/Çıkış fonksiyonları Tablo-18.2'de verilmiştir.

Komut	İşlem
getchar()	Klavyeden bir karakter okur ve ENTER tuşuna basılmasını bekler.
getch()	Klavyeden bir karakter okur ve ENTER tuşuna basılmaksızın okunan karakteri bir değişkene aktarır. Klavyeden girilen karakter ekranda görüntülenmez.
putc(), putchar()	Ekrana bir karakter yazar.
gets()	Klavyeden girilen bir stringi (kelime dizini) bir değişkene aktarır.
puts()	Bir stringi ekrana yazar.
printf()	String ifadelerin belli bir formatta gönderilmesini sağlar.
KBHIT()	RS232 veri alma kontrol fonksiyonu

Tablo-18.2. *CCS C derleyicisi RS232 Giriş/Çıkış fonksiyonları*

GETC( ), GETCH( ), GETCHAR( ), FGETCH( )

Bu fonksiyonlar RS232 ara yüzünün alma bit'inden (RX) bir karakterin gelmesini bekler, karakter geldiğinde o karakterin değeri ile geri döner. Geri dönüş değeri 8 bit'lik bir değerdir.

***değişken ismi* = getc( );**
***değişken ismi* = getch( );**
***değişken ismi* = getchar( );**
***değişken ismi* = fgetch( );**

PUTC( ), PUTCHAR( ), FPUTC( )

Bu fonksiyonlar, RS232 gönderme bit'i üzerinden (TX) bir karakter gönderme işini yaparlar. Fonksiyonlardaki *"data"* kısmı 8 bit'liktir.

putc(*data*);
putchar(*data*);
fputc(*data, sabit byte değer*);

GETS( )
Bu fonksiyon string (bir karakterden fazla olan ifadeler) ifadelerin RS232 üzerinden alınmasını sağlar.

gets(*değişken ismi*);

```
char sifre[20]; // 20 karakterlik sifre isminde bir değişken tanımlanıyor.
printf("Sifre Gir:  ");
gets(sifre);
```

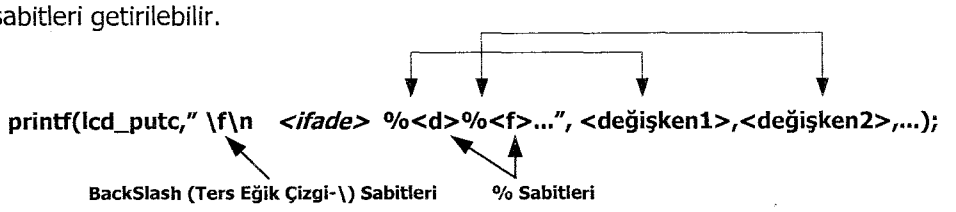
Örnekte RS232 hattından gelen klavyeden yazılan karakterler bir string olarak sifre adlı değişkene aktarılıyor.

PUTS( )
RS232 üzerinden string ifadelerin gönderilmesini sağlar.

puts(*string*);

```
puts(" -------Step Motor Kontrol Programı--------");
```

PRINTF( )
Bu fonksiyon string ifadelerin belli bir formatta gönderilmesini sağlar. Fonksiyon genel olarak aşağıda verilen formatta kullanılır. Fonksiyonda "\" kısmına Bölüm 4.6.1'de verilen backslash sabitleri getirilebilir. Fonksiyondaki "%" kısmı ile gönderilecek değişkenin formatı tanımlanır. Bu kısma da Bölüm 4.6.2'de verilen % sabitleri getirilebilir.



KBHIT( )
Bu fonksiyon, RS232 biriminin alma bit'inden (RX) alınacak karakterin başlama bit'inin gelip gelmediğini kontrol eder. Böylece alınacak bilgi için kullanıcı ne zaman işlem yapıp yapmayacağını ayarlar. Eğer gelecek karakterin başlama bit'i RX pin'ine gelmişse fonksiyon TRUE (1) bilgisi ile eğer başlama bit'i gelmemişse FALSE (0) bilgisi ile geri döner.

***değişken ismi* =kbhit( );**

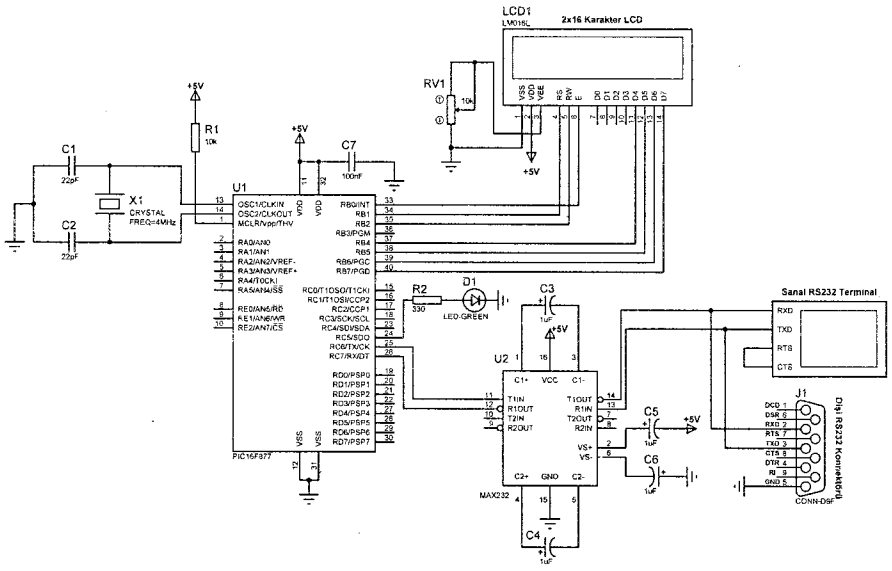
18.7. RS232 SERİ İLETİŞİM UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada klavyeden girilen bilgilerin LCD ekranında görüntülenmesi sağlanmıştır. Devrede gönderilen verilerin karşı taraftan doğru alınıp alınmadığının kontrolü yapılmak istenmediği için sadece TXD, RXD ve GND uçları kullanılmıştır.

Diğer uçlar şekilde görüldüğü gibi birbirine bağlanmıştır. Devrede 9 pin'li dişi RS232 port'unun bağlantısı verilmiştir. Programda RS232 port'undan veri alındığında kesme meydana gelmesi sağlanmıştır. **İlgili program kodları kesme fonksiyonuna yazılmıştır. Böylece RS232 port'undan veri alınmadığı zamanlarda denetleyiciye başka işlemler yaptırabilirsiniz.** Ayrıca her kesme meydana geldiğinde RC5 pin'ine bağlı LED'in yanıp sönmesi sağlanmıştır. Böylece verinin geldiği LED ile de görsel olarak gösterilmiş olmaktadır.

Devreyi benzetim programında çalıştırdığımızda seri iletişimi görsel olarak görebilmek için Proteus-ISIS programında bulunan sanal RS232 Terminali (Virtual Terminal) devreye eklenmiştir. Böylece sanal olarak seri iletişim bağlantısı yapmış olacağız. Pratikte devreyi gerçekleştirirken bu sanal terminal devreye eklenmeyecektir. **Proteus programında sanal RS232 Terminali (Virtual Terminal) üzerine fare ile sağ tuş ardından sol tuş ile tıklanarak açılan kontrol menüsünde "Advanced Properties" kısmında "RX/TX Polarity" ayarının "Inverted" olarak seçildiğinden emin olunuz.** Gerçekte devrenizi denemek için Windows işletim sisteminde bulunan **HyperTerminal** programı (Başlat-Programlar-Donatılar-HyperTerminal) veya CCS C programında bulunan **siow.exe** programı (CCS C programında "Tools" menüsünde "Serial Port Monitor" komutu seçilerek veya C:\Program Files\PICC\siow.exe dizinine girilerek) çalıştırılabilir. Bu programları kullandığınızda, programınızda kullandığınız iletişim hızı, stop bit'i, eşlik bit'i ve bilgisayarın kullandığınız port ismi (COM1, COM2...) gibi ayarları doğru bir şekilde yapmayı unutmayınız. RS232 port'undan gönderdiğiniz bilgiler içinde Türkçe karakterler kullanmamanız gerekmektedir. Türkçe karakterler kullanırsanız, bu karakterler yerine başka değişik karakterler gözükecektir.



Şekil-18.8. *RS232 Seri iletişim uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile RS232 Seri İletişim Uygulaması
PIC PROG/DEKA     : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#define use_portb_lcd TRUE   // LCD B port'una bağlı
#include <lcd.c>   // lcd.c dosyası tanıtılıyor

#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)
// RS232 iletişim ayarları belirtiliyor

char   klavye[80];  // klavye isminde 80 üyeli bir dizi tanımlanıyor

#int_rda   // RX ucuna veri gelince meydana gelen kesme
void serihaberlesme_kesmesi ()
{
   disable_interrupts(int_rda); // int_rda kesmesini pasif yap
   output_high(pin_c5);  // RC5 çıkışı lojik-1
   gets(klavye);//Metni al ve "klavye" adlı dizi değişkenine aktar.
   printf("\n\rYazdiginiz Metin>  %s\n",klavye); // Satır atla, yeni
//satır, RS232 üzerinden klavye string ifadesini gönder,satır atla
   printf(lcd_putc,"\f%s",klavye);   //LCD'yi  temizle  ve  klavye
                                     // değişkenini LCD'de göster
   output_low(pin_c5); // RC5 çıkışı lojik-0
   printf("\n\rKlavyeden  bir  metin  giriniz  ve  enter  tusuna
basiniz>");
// Satır atla, yeni satır,RS232 üzerinden belirtilen metni gönder
}

/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   output_low(pin_c5); // RC5 çıkışı ilk anda sıfırlanıyor
   lcd_init();   // LCD'yi hazırla
```

```
   printf("\r**************** Merhaba ********************");  //
Yeni satır, RS232 üzerinden belirtilen metni gönder ve satır atla.
   printf("\n\rKlavyeden girdiginiz bilgiler LCD'de
gorunecektir\n\n"); // Satır atla, yeni satır, RS232 üzerinden
                    // belirtilen metni gönder
   printf("\n\rKlavyeden bir metin giriniz ve enter tusuna
basiniz>"); // Satır atla, yeni satır,RS232 üzerinden belirtilen
            //metni gönder
   enable_interrupts(GLOBAL);// Aktif edilen tüm kesmelere izin ver

   while(1) // Sonsuz döngü
   {
      enable_interrupts(int_rda); // int_rda kesmesi aktif
   }
}
//*******************************************************************
```

Programda `#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)` komutu ile RS232 iletişim hızının 9600 olduğu, veri gönderme bit'inin RC6 pin'i, veri alma pin'inin ise RC7 pin'i olduğu, eşlik bit'inin olmadığı ve stop bit'inin 1 bit olacağı belirtilmiştir. `char klavye[80];` komutu ile 80 elemanlı bir dizi tanımlanmıştır. Bilgisayar klavyesinden gelecek bilgi bu dizide saklanmaktadır.

`#int_rda` ile RS232 hattından bilgi geldiğinde oluşacak kesmenin fonksiyonu tanımlanmıştır. Bu fonksiyonda ilk başta `disable_interrupts(int_rda);` komutu ile kesme bayrağı sıfırlanmıştır. RS232 kesmesinde bu işlemin yapılması mecburidir. Programda `printf("\n\rKlavyeden bir metin giriniz ve enter tusuna basiniz>");` komutu ile RS232 port'una veri gönderilmiştir. Komutta `\n` ile satır atlanmış, `\r` ile yeni satıra geçilmiştir ve ardından `"Klavyeden bir metin giriniz ve enter tusuna basiniz>"` bilgisi RS232 port'una gönderilmiştir. Kesme fonksiyonunda `gets(klavye);` komutu ile RS232 port'undan gelen bilgi alınmaktadır. `Gets()` fonksiyonu RS232 port'undan veri gelene kadar ve bekler. `printf("\n\rYazdiginiz Metin> %s\n",klavye);` komutu ile de `\n` ile satır atlanmış, `\r` ile yeni satıra geçilmiştir ve ardından `"Yazdiginiz Metin>"` bilgisi RS232 port'una gönderilmiştir. Ardından komuttaki %s kısmına string formatında "klavye" dizi değişkeni içeriği gönderilmiş ve `\n` ile de en son olarak satır atlatılmıştır. `printf(lcd_putc,"\f%s",klavye);` komutu ile de LCD'ye seri port'tan gelen bilgi string formatında yazdırılmıştır. Sonsuz döngüde `enable_interrupts(int_rda);` komutu ile her kesme çıkışında pasif edilen kesme yeniden aktif edilmiştir.

Program sadece RS232 port'undan veri geldiğinde kesme fonksiyonundaki işlemleri yapacaktır. Onun haricinden ana programda sonsuz döngü içine başka komutlar yazılarak, RS232 hattından bilgi gelmediği süre boyunca başka işlemler de yaptırılabilir.

19. Bölüm

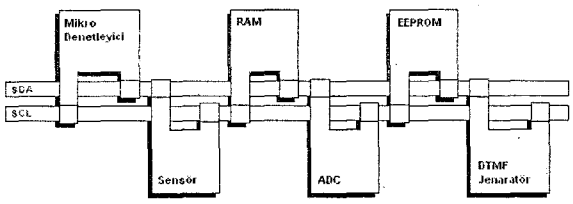
I²C İLE SPI SERİ İLETİŞİM VE HARİCİ BELLEK İŞLEMLERİ

- I²C Seri İletişim
- #USE I²C() Fonksiyonu
- I²C_START Fonksiyonu
- I²C_WRITE Fonksiyonu
- I²C_READ Fonksiyonu
- I²C_STOP Fonksiyonu
- I²C_POLL Fonksiyonu
- CCS C Harici EEPROM Kütüphaneleri
- 24C02 Harici EEPROM
- CCS C Dosyalarını Kullanarak 24C02 Harici EEPROM Uygulaması
- Kendi Harici EEPROM Fonksiyonlarımızı Yazma ve Uygulaması
- SPI İletişim
- SETUP_SPI() Fonksiyonu
- SPI_WRITE() Fonksiyonu
- SPI_READ() Fonksiyonu
- SPI_DATA_IS_IN()

Bu bölümde **I²C ve SPI** seri iletişim metotları hakkında bilgi verilmiş ardından CCS C derleyicisinde I²C ve SPI iletişimlerinin hangi fonksiyonlarla nasıl sağlandığı incelenmiştir. Daha sonra ise I²C ve SPI seri iletişim metotları ile kontrol edilen harici EEPROM'lar için CCS C kütüphanelerinde bulunan dosyalar ve fonksiyonları tanıtılmıştır. Harici EEPROM olarak da Microchip firmasının **24C02 EEPROM** entegresi tanıtılmıştır. Diğer 24XX entegreleri tanıtılan 24C02 entegresi ile çok benzerdir. Anlatılan bilgilerle kolayca diğer entegreleri de kullanabilirsiniz. CCS C derleyicisi kütüphanelerinde bulunan hazır harici EEPROM dosyasını kullanarak 24C02 harici EEPROM uygulaması gerçekleştirilmiştir. En son olarak da kendi harici EEPROM kontrol fonksiyonlarımızın oluşturulması anlatılmış ve aynı uygulamada denenmiştir. Böylece entegrelerin kontrolü için hazır kütüphaneleri kullanmanın yanında istediğimiz entegreyi kontrol etmek için kendi fonksiyonlarımızı oluşturma becerisini de kazanmış olmanız sağlanmaktadır.

19.1. I²C SERİ İLETİŞİM

I²C (**I**nter-**I**ntegrated **C**ircuit) seri iletişimi, mikrodenetleyicinin diğer entegre elemanları ile kolayca iletişim kurmasını sağlayan entegreler arası bir seri haberleşme protokolüdür. Bu protokol **PHILIPS** firması tarafından geliştirilmiştir. **Veri alma ve gönderme hızının yüksek olunmasının gerekmediği, veri giriş/çıkış bacak sayısının önemli olduğu uygulamalarda** en az sayıda hat ile iletişim gereksinimi ortaya çıkmıştır. Bu nedenle I²C ve SPI gibi çeşitli iletişim protokolleri geliştirilmiştir. Bazı PIC denetleyicileri içinde hem I²C hem de SPI iletişim metotlarını içinde barındıran MSSP (The **M**aster **S**ynchronous **S**erial **P**ort) modülü bulunmaktadır. Şekil-19.1'de mikrodenetleyici ile I²C iletişimini kullanan elemanların bağlantısı görülmektedir.



Şekil-19.1. *Elemanların mikrodenetleyiciye I²C bağlantısı.*

I²C protokolünde, **seri veri giriş-çıkış (SDA)** ve **seri saat sinyali (clock) (SCL)** uçları ile iletişim sağlanmaktadır. SDA bilgi giriş çıkış işlemleri SCL ucundaki saat sinyali sayesinde senkronize bir şekilde gerçekleşmektedir. Buradan da anlaşıldığı gibi **I²C protokolü sadece 2 hat** üzerinden yapılmaktadır. Bu protokolde entegreler ya **master (ana)** ya da **slave (uydu)** olabilir. Entegre **master** iken, master olan entegre kendi gönderdiği saat sinyalleri ile iletişimi kontrol eder konumdadır. Slave konumunda olan entegre saat sinyali üretmez sadece veri alış veriş işlemi yapar. I²C protokolü **Yavaş (slow), Hızlı (fast) ve Yüksek hızlı (High Speed)** olmak üzere çeşitli hızlarda veri iletişimi sağlayabilir. Standart yani yavaş hızda veri iletişim hızı **100 kbit/s**'e kadar, hızlı (fast mode) iletişimde veri iletişim hızı **400 kbit/s**'e kadar, yüksek hızlı iletişimde ise veri iletişim hızı **3.4 Mbit/s**'e kadar çıkabilmektedir. Bu hızlar denetleyici içinde bulunan I²C donanımına bağlıdır. Şekil-19.2'de I²C iletişiminin komple bir veri gönderme işleminin sinyal olarak gösterimi görülmektedir. I²C iletişiminde veri göndermek için ilk başta **başlama bit'i** gönderilir, ardından **adres bilgisi** ve **veri bit'leri** gönderilir. Veri gönderimi **stop bit'i** ile sonlanır. Her bir byte bilgi gönderimine karşılık karşı taraf, bir alındı bit'i (**ACK**) gönderir. İletişim için gerekli durumları şöyle özetleyebiliriz.

Yol Meşgul Değil seviyesinde = Veri (SDA) ve Clock (SCL) hatlarının ikisi de lojik-1 olmalıdır.

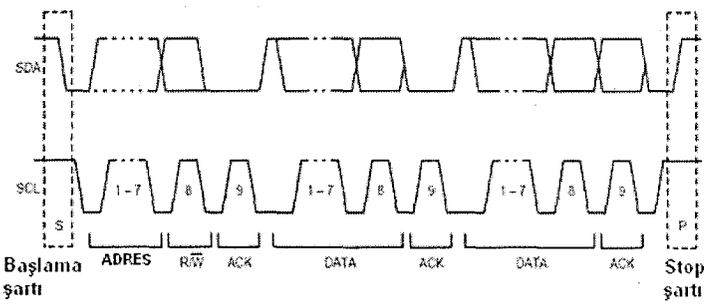
Veri Transferine Başla = Clock hattı (SCL) lojik-1 iken, veri hattı (SDA) lojik-1 seviyesinden lojik-0 seviyesine getirilir. Bu durumda veri transferine başlanmış olunur.

Veri Transferini Durdur = Clock hattı (SCL) lojik-1 iken, veri hattı (SDA) lojik-0 seviyesinden lojik-1 seviyesine getirilir. Bu durumda veri transferi durdurulmuş olur.

Geçerli Veri = Her veri gönderimi başla bit'i ile (start) başlar ve dur bit'i (stop) ile son bulur. İletişim sırasında gönderilen veri, clock hattı (SCL) lojik-0 durumunda iken değiştirilmelidir. Başla ve Dur bit'leri arasında istenildiği kadar veri gönderilebilir. Her bilgi byte olarak (8 bit) gönderilir. Her 9. bit ise karşı taraftan gelen alındı (ACK) bit'idir.

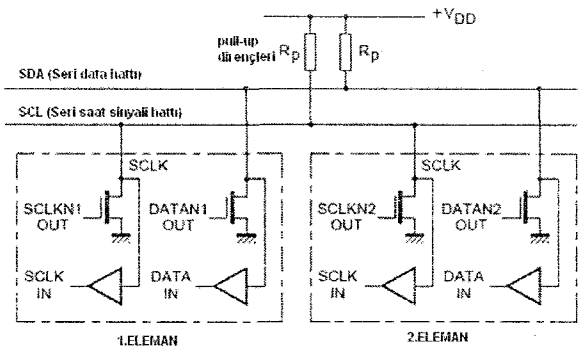
I^2C iletişiminde master (ana) olan eleman başlama bit'inden (start) sonra alıcı yani slave (uydu) konumundaki elemana ilk başta adres byte'ı gönderir. Daha sonra ise veri byte'larını gönderir. Alıcı (slave) eleman her aldığı byte'a karşılık bir **alındı (ACK)** bit'i gönderir. Master eleman tarafından gönderilen adres bilgisi slave (uydu) konumundaki cihazın adres bilgisidir. Böylece master eleman hangi alıcı elemana bilgi gönderdiğini belirtmiş olur. Çünkü aynı anda birden fazla alıcı (slave) eleman olabilir.

Slave (uydu) elemanı master (ana) elemana yine byte olarak veri gönderir ve master elemanı her aldığı byte için bir kabul bit'i gönderir. En son alınan byte'tan sonra alındı işareti gönderilmez.



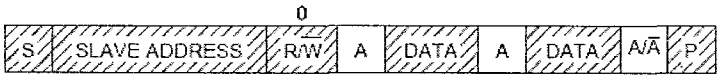
Şekil-19.2. *I^2C iletişiminde veri gönderme işlemi.*

I^2C iletişiminde kullanılan **SCL** ve **SDA** uçlarının iç yapısı Şekil-19.3'de görülmektedir. Şekilden de görüleceği üzere **SCL ve SDA hatları pull-up direnci ile V_{DD} hattına bağlanmalıdır.**



Şekil-19.3. *I²C iletişimi bağlantıları.*

I²C iletişiminde **master (ana) elemanı slave (uydu) elemanına bir veri gönderme işleminde** start bit'inden sonra ilk başta veri göndereçeği slave elemanının adresini 7 bit olarak gönderir, 8. bit ise R/W bit'i olan okuma yazma bit'idir. Bu bit "0" ise yazma işleminin yapılacağı, "1" ise okuma işleminin yapılacağını bildirir. Bunun sonucunda slave olan eleman bir alındı (ACK) bit'i gönderir. Bunun ardından master olan eleman 8 bit'lik (1 byte'lık) veri gönderir ve bunun ardından karşı taraftan tekrar bir alındı bit'i alır. Bu işlemler master elemandan gönderilen dur bit'ine kadar devam eder. Şekil-19.4'de anlatılan bu işlem daha net görülmektedir. **Master olan eleman slave olan elemandan veri okurken ise**, başla bit'inden sonra master olan eleman slave olan elemanın adres bilgisini gönderir ve ardından R/W bit'ini "1" olarak gönderir. Ardından slave olan elemandan alındı bit'i gelir ve slave olan eleman hemen 8 bit'lik data (bir byte'lık) gönderir. Slave olan elemandan bir byte'lık veri alındığında master olan eleman karşı tarafa bir bit alındı bit'i gönderir. Bu işlemler master olan elemanın dur bit'ini gönderene kadar devam eder. Şekil-19.5'de anlatılan bu işlemler daha net anlaşılmaktadır.



Şekil-19.4. *Master olan elemandan slave olan elemana veri gönderme işlemi.*

Şekil-19.5. *Master olan elemanın slave olan elemandan veri okuma işlemi.*

▨ Master olan elemandan slave olan elemana
□ Slave olan elemandan master olan elemana

S = Başlama (start) bit'i
SLAVE ADDRESS = Slave olan elemanın adresi
R/W = Okuma-Yazma bit'i. "1" ise okuma, "0" ise yazma
A = Slave olan elemandan gelen alındı (ACK) biti
DATA = Gönderilmek istenen 8 bit'lik veri
P = Dur (stop) bit'i

19.1.1. #USE I²C () Fonksiyonu

Bu komut derleyiciye I²C iletişim protokolünün ayarlarını belirtir. Bu ayarlar, I²C iletişimde kullanılacak bit'lerin isimleri, denetleyicinin Master veya Slave olup olmadığı gibi bilgilerdir. Fonksiyonda *"sabit"* kısımlarına aşağıda verilen sabit tanımlamalar yazılabilir.

#use i2c(*sabit, sabit, sabit,...*)

#USE I²C () Fonksiyonu Sabitleri

multi_mode = Çoklu master modu.
slave = I²C slave modunda
master = I²C master modunda
SCL=pin I²C iletişiminde SCL pin'ini belirler. (SCL=pin_B0 gibi)
SDA=pin I²C iletişiminde SDA pin'ini belirler. (SDA=pin_B1 gibi)
Slow = I²C iletişiminde iletişimin hızının yavaş olduğunu belirler. İstenirse iletişim hızının değeri de belirtilebilir. (slow=100000 gibi)
Fast = I²C iletişiminde iletişimin hızının hızlı olduğunu belirler. İstenirse iletişim hızının değeri de belirtilebilir. (fast=400000 gibi)
Address = değer Slave modunda slave mod adres değerini belirler.(address=0x30 gibi)
RESTART_WDT = i2c_read() fonksiyonu veri okumayı beklerken WDT'ı sıfırlayarak, WDT'ın denetleyiciyi resetlemesini önler.

Örnek:#use i2c(master, sda=pin_C4, scl=pin_C3, fast=400000)

19.1.2. I²C_START(), I²C_WRITE(), I²C_READ(), I²C_STOP(), I²C_POLL() Fonksiyonları

Bu fonksiyonlar I²C seri iletişiminde; iletişime başlamayı, veri alma, veri gönderme ve iletişimi bitirmeyi sağlama gibi işlemlerin yapılmasını sağlayan fonksiyonlardır. Bu fonksiyonların kullanılabilmesi için programın başında **#USE I²C()** ön işlemci fonksiyonunun tanıtılması gerekmektedir.

I2C_START( )
I²C iletişimi için başlama fonksiyonudur (I²C master modunda iken). I²C iletişimine başlamak için ilk başta bu fonksiyon çağrılmalıdır.
I2C_WRITE(data)
I²C iletişimi üzerinden bir byte veri gönderir. Fonksiyondaki *"data"* kısmına 8 bit'lik bir veri yazılır.
I2C_READ( )
I²C iletişimi üzerinden bir byte'lık veri okunmasını sağlar.
I2C_STOP( )
I²C iletişimini durdurma fonksiyonudur (I²C master modunda iken). I²C iletişimine başlamak için ilk başta bu fonksiyon çağrılmalıdır.

I2C_POLL( )

I^2C donanımı tampon (buffer) biriminden bir byte'lık veri aldığında bu fonksiyon 1 (TRUE) bilgisi ile geri döner. Eğer tamponda bilgi yoksa fonksiyon 0 (FALSE) bilgisi ile geri döner.

Örnek I^2C iletişimi;

```
i2c_start();        // I²C İletişimi başlat
i2c_write(0xa0); // Slave konumundaki cihaz adresi
i2c_write(5);       // Komut gönderiliyor
i2c_write(12);      // Data (veri) gönderiliyor
i2c_stop();         // I²C iletişimi durduruluyor
```

19.1.3. CCS C Harici EEPROM Kütüphaneleri

CCS C derleyicisi içinde çeşitli harici EEPROM entegreleri için yazılmış kütüphane dosyaları mevcuttur. Bu dosyaları kullanarak kolay bir şekilde bu entegreleri programınızda kullanabilirsiniz. Bu dosyalar C:\Program Files\PICC\Drivers dizini içinde bulunmaktadır. Bu dosyalardan bazılarının isimleri ve açıklamaları aşağıda verilmiştir.

2401.c → Microchip firmasının 24LC01B seri EEPROM entegresi fonksiyonları.
2402.c → Microchip firmasının 24LC02B seri EEPROM entegresi fonksiyonları.
2404.c → Microchip firmasının 24LC04B seri EEPROM entegresi fonksiyonları.
2408.c → Microchip firmasının 24LC08B seri EEPROM entegresi fonksiyonları.
2416.c → Microchip firmasının 24LC16B seri EEPROM entegresi fonksiyonları.
2432.c → Microchip firmasının 24C32B seri EEPROM entegresi fonksiyonları.
2464.c → Microchip firmasının 24LC64 seri EEPROM entegresi fonksiyonları.
24128.c → Microchip firmasının 24C128 seri EEPROM entegresi fonksiyonları.
24256.c → Microchip firmasının 24LC256 seri EEPROM entegresi fonksiyonları.
24512.c → Microchip firmasının 24LC512 seri EEPROM entegresi fonksiyonları.
24515.c → Microchip firmasının 24FC515 seri EEPROM entegresi fonksiyonları.
25040.c → Microchip firmasının 25C040 seri EEPROM entegresi fonksiyonları.
25080.c → Microchip firmasının 25C080 seri EEPROM entegresi fonksiyonları.
25160.c → Microchip firmasının 25C160 seri EEPROM entegresi fonksiyonları.
25320.c → Microchip firmasının 25C320 seri EEPROM entegresi fonksiyonları.
25640.c → Microchip firmasının 25C640 seri EEPROM entegresi fonksiyonları.
25c080.c → Microchip firmasının 25C080 seri EEPROM entegresi fonksiyonları.
9306.c → Microchip firmasının 93CS06LN seri EEPROM entegresi fonksiyonları.
9313.c → Microchip firmasının 93C13N seri EEPROM entegresi fonksiyonları.
9346.c → Microchip firmasının 93AA46 seri EEPROM entegresi fonksiyonları.
9356.c → Microchip firmasının 93C56 seri EEPROM entegresi fonksiyonları.
9366.c → Microchip firmasının 93C66 seri EEPROM entegresi fonksiyonları.
9376.c → Microchip firmasının 93AA76C seri EEPROM entegresi fonksiyonları.
9386.c → Microchip firmasının 93AA86C seri EEPROM entegresi fonksiyonları.

9512.c → Microchip firmasının 95C12N seri EEPROM entegresi fonksiyonları.
AT2421.c → ATMEL firmasının 24C21 seri EEPROM entegresi fonksiyonları.
AT25256.c → ATMEL firmasının 24128 ve 24256 seri EEPROM entegreleri fonksiyonları.
ds2432.c → Maxim-Dallas firmasının DS2432 EEPROM entegresi fonksiyonları.

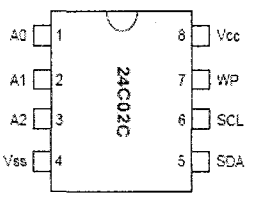
Yukarıda verilen dosyaların içerdikleri fonksiyonlar genelde aynıdır. Aşağıda bu fonksiyonlar ve görevleri açıklanmıştır.

init_ext_eeprom(); Diğer harici EEPROM fonksiyonları çağrılmadan bu fonksiyon çağrılmalıdır.
write_ext_eeprom(*adres, değer*); Harici EEPROM'a "adres" değişkeni ile belirtilen adrese, byte olarak belirtilen "değer" değişkeni değerini yazar.
***değişken ismi* = read_ext_eeprom(*adres*);** Harici EEPROM'dan "adres" değişkeni ile belirtilen adresteki veriyi okur.
***değişken ismi* = ext_eeprom_ready(*adres*);** EEPROM'un işlem kodu almaya hazır olup olmadığını anlamaya yarar. EEPROM işlem kodu almaya hazır ise 1 (TRUE) değeri ile, hazır değilse 0 (FALSE) değeri ile geri döner.

NOT= Bu dosyaların içinde kullanılacak denetleyici uçları tanımlanmıştır. Örneğin SDA ve SCL bit'lerinin PIC'in hangi uçları olacağı bu dosyalarda belirtilmiştir. Ayrıca diğer I²C ayarları için de bu dosyalar içinde #use i2c() fonksiyonu kullanılmıştır. Bu fonksiyon eğer ana programda belirtilmemişse, bu dosyalar içindeki ayarlar geçerli olur.

19.1.4. 24C02 EEPROM Entegresi

24C02 entegresi Microchip firmasının 2K I²C Seri EEPROM entegresidir. 2K ifadesi 2048 bit anlamındadır. Bu da bize bu EEPROM entegresinin 256 Byte (256x8 bit = 2048 bit) kapasitede olduğunu bildirir. Aşağıda Şekil-19.6'da 24C02C EEPROM entegresinin pin diyagramı görülmektedir.



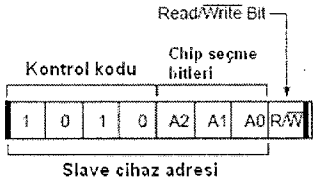
Şekil-19.6. *24C02C EEPROM entegresinin pin diyagramı.*

Pin açıklamaları:
WP: Donanım yazma koruması. **A0, A1, A2 :** Çip seçme uçları.
SCL: Saat sinyali (clock). **Vcc:** + Besleme, **Vss:** Şase.
SDA: Seri Veri ucu (data). **NC:** Boş uç (No Connection).

24C02C entegresi 100Khz ve 400Khz uyumludur. 3 bit'lik adres hattı ile 8 adet EEPROM adreslenebilir yani aynı iletişim hattında bulunabilir. Toplam 1.000.000 kez

silme/yazma işlemi yapılabilir. 200 yıldan fazla bir süre içine yazılan bilgileri tutabilir.

24C02C EEPROM'u ile I^2C protokolü ile iletişime geçmeden önce I^2C protokolü "I2C_START();" fonksiyonu kullanarak başlatılmalıdır. Daha sonra EEPROM'a kontrol bilgisi "I2C_WRITE(data);" fonksiyonu kullanılarak gönderilmelidir. **Kontrol bilgisi aşağıda gösterilen bit'lerden oluşur.**



Şekil-19.7. *Kontrol bilgisi bit'leri.*

Kontrol bilgisi bit'lerinde yüksek seviyeli 4 bit 24C02C EEPROM entegresi için her zaman heksadesimal olarak A (desimal 10) değerindedir. A2, A1, A0 bit'leri ise slave konumunda olan entegre adres bilgileridir. 3 bit ile 8 adet slave konumunda entegte adreslenebilir ve aynı veri hattı ile bu entegreler ile iletişim kurulabilir. Örneğin bir entegrenin A2 ucu toprağa, A1 ucu toğrağa ve A0 ucu ise + beslemeye bağlanmış ise bu entegrenin adresi (A2A1A0 = 001) olur. Kontrol bilgisinin en sonunda yer alan R/$\overline{W}$ bit'i ise EEPROM'a yazma mı? yoksa okuma mı? yapılacağını belirler. R/$\overline{W}$ bit'i 0 ise yazma işlemi 1 ise okuma işlemi yapılır.

EEPROM entegresinde bulunan WP pin'i ise istenirse yazmaya karşı koruma sağlar. Bu pin lojik-1 olursa (Vcc'ye bağlanırsa) 080-0FF arası adres alanları yazmaya karşı korunur. WP ucu lojik-0 olursa (Vss'ye bağlanırsa) korumanın olmadığı, yani tüm adreslere veri yazılabileceği anlamına gelir.

19.1.5. I^2C ve SPI Kesmesi

I^2C ve SPI seri iletişimlerinde herhangi birktivite başladığında kesme meydana getirilebilir. PIC14XX serisi denetleyicilerde I^2C iletişimi kesmesi için **#INT_I2C** kesmesi kullanılır. Diğer PIC denetleyicilerde ise hem I^2C hem de SPI seri iletişim kesmeleri için **#INT_SSP** kesmesi kullanılır.

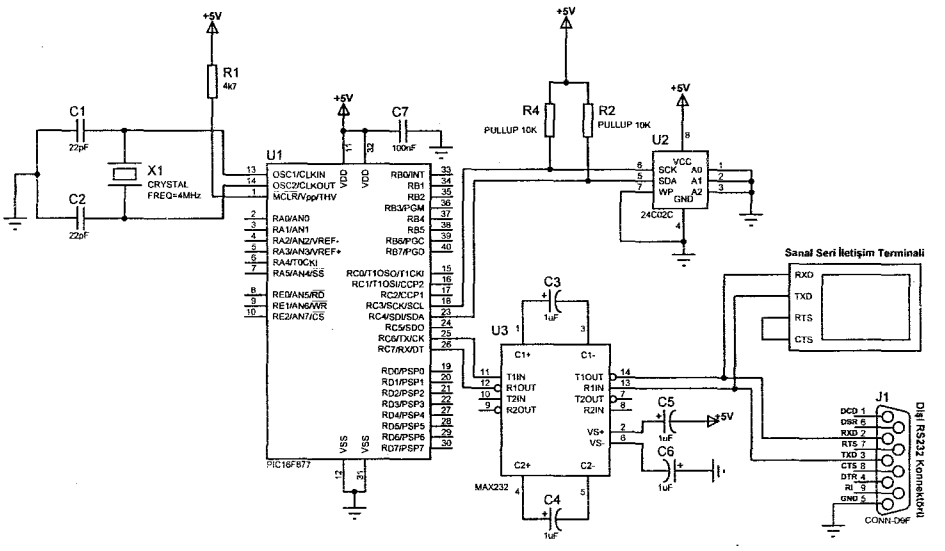
19.1.6. CCS C Dosyalarını Kullanarak 24C02 Harici EEPROM Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada I^2C seri iletişim protokolü kullanan Microchip firmasının 24C02 2K (256 Byte) entegresi kullanılmıştır. RS232 iletişimi ile bilgisayar ile 24C02 EEPROM entegresinin istenen adresine veri yazımı ve istenen adresinden veri okuma işlemi

yapılmıştır. I²C iletişimi için SDA ucu olarak RC4 pin'i, SCK ucu olarak da RC3 pin'i kullanılmıştır. Siz kendiniz #use i2c() fonksiyonunu değiştirerek iletişim uçlarını farklı pin'lerle yapabilirsiniz. Programda 24C02 entegresine veri yazma ve okuma işlemleri için CCS C kütüphanesinde bulunan hazır fonksiyonlar içeren 2402.c dosyası kullanılmıştır. **24C02 entegresinde iletişim kurmak için iletişimde kullanılan veriler heksadesimal formatta olması gereklidir. Byte olarak RS232 port'undan alınan verilerin heksadesimal formata çevirmek için de CCS C kütüphanesinde bulunan input.c dosyası programa eklenmiştir. Bu dosyadaki gethex(); fonksiyonu seri port'tan veriyi alır ve heksadesimal formatına çevirir.**

Devreyi benzetim programında çalıştırdığımızda seri iletişimi görsel olarak görebilmek için Proteus programında bulunan sanal RS232 Terminal (Virtual Terminal) devreye eklenmiştir. Böylece sanal olarak seri iletişim bağlantısı yapmış olacağız. Pratikte devreyi gerçekleştirirken bu sanal terminal devreye eklenmeyecektir. **Proteus programında sanal RS232 Terminali (Virtual Terminal) üzerine fare ile sağ tuş ardından sol tuş ile tıklanarak açılan kontrol menüsünde "Advanced Properties" kısmında "RX/TX Polarity" ayarının "Inverted" olarak seçildiğinden emin olunuz.** Gerçekte devrenizi denemek için Windows işletim sisteminde bulunan **HyperTerminal** programı (Başlat-Programlar-Donatılar-HyperTerminal) veya CCS C programında bulunan **siow.exe** programı (CCS C programında "Tools" menüsünde "Serial Port Monitor" komutu seçilerek veya C:\Program Files\PICC\siow.exe dizinine girilerek) çalıştırılabilir. Bu programları kullandığınızda, programınızda kullandığınız iletişim hızı, stop bit'i, eşlik bit'i ve bilgisayarın kullandığınız port ismi (COM1, COM2...) gibi ayarları doğru bir şekilde yapmayı unutmayınız.



Şekil-19.8. *CCS C dosyalarını kullanarak 24C02 EEPROM uygulaması devre şeması*

Devre programı;

```
/*****************************************************************
 PIC16F877 ile CCS C Dosyalarını Kullanarak 24C02 EEPROM Uygulaması
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları
#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              // osilatör frekansı belirtiliyor
#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)
// RS232 ayarları belirtiliyor

#use i2c(master,sda=pin_c4,scl=pin_c3,slow=100000)//I²C Fonksiyonu,
             // SDA ve SCL pin'leri ve iletişim hızı belirleniyor
#include <input.c> // input.c dosyası programa ekleniyor
#include <2402.c>  // 2402.c dosyası programa ekleniyor

byte veri,adres; // byte tipinde değişken tanımlanıyor
char islem;      // char tipinde değişken tanımlanıyor

//******************* ANA PROGRAM FONKSİYONU *******************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   printf("\n\r*******************************************************");
   printf("\n\r  24C02C Harici EEPROM Bilgi Okuma Yazma  ");
   printf("\n\r*******************************************************");
   printf("\n\n\r Bir islem seciniz>");

   while(1)
   {
      do  // do-while döngüsü tanımlanıyor
      {
         printf("\n\rOkuma (O) veya Yazma (Y) >");
         islem=getc(); // RS232 port'undan gelen veri alınıyor
         islem=toupper(islem); //  toupper  fonksiyonu  karakterleri
                               // büyük karaktere çevirir (a>A)
         putc(islem);   //"islem" değişkeni içeriği RS232 port'una
                        // gönderiliyor
      }  while ( (islem!='O') && (islem!='Y') ); // Girilen karakter
                        // "O" veya "Y" dışında ise döngü başına dön
```

```
        if (islem=='O')  // Eğer okunan karakter "O" ise
        {
           printf("\n\rOkumak istediginiz adresi giriniz> ");
           adres=gethex(); // RS232 port'undan gelen veriyi
                           // heksadesimal formatında al
           printf("\n\rDeger= %X",read_ext_eeprom(adres) ); // "adres"
         // değişkeninin tuttuğu adresteki bilgiyi oku RS232'ye gönder
        }

        if (islem=='Y')  // Eğer okunan karakter "Y" ise
        {
           printf("\n\rBilgi yazmak istediginiz adresi giriniz> ");
           adres=gethex(); // RS232 port'undan gelen veriyi
                           // heksadesimal formatında al
           printf("\n\rBilgiyi giriniz> ");
           veri=gethex(); // RS232 port'undan gelen veriyi
                          // heksadesimal formatında al
           write_ext_eeprom(adres,veri); // Harici EEPROM'da
                            // belirtilen adrese "veri" değerini yaz
        }
    }
}
//*****************************************************************
```

Programda `#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)` komutu ile RS232 seri iletişim ayarları belirtilmiştir. Bu komutta RS232 iletişim hızının 9600 olacağı, veri gönderme bit'inin RC6, veri alma bit'inin RC7 olduğu, eşlik bit'inin olmadığı ve stop bit'inin 1 bit olacağı belirtilmiştir. `#use i2c(master,sda=pin_c4,scl=pin_c3,slow=100000)` komutu ile de I²C seri iletişim ayarları belirtilmiştir. Bu komut ile I²C seri iletişiminde denetleyicinin master konumda olduğu, SDA hattının RC4 pin'i, SCL hattının RC3 pin'i olduğu ve iletişim hızının yavaş olarak 100 Kbit/s olarak gerçekleştirilmesi belirtilmiştir. `#include <input.c>` komutu ile seri iletişimlerde alınan verilerin heksadesimal olarak alınmasını sağlayan fonksiyonları içeren input.c dosyası programa eklenmiştir. Bu dosyanın içindeki gethex() komutu ile RS232 port'undan veri alınır ve heksadesimal formata çevrilir. `#include <2402.c>` komutu ile de 24C02 EEPROM entegresi kontrol fonksiyonlarını içeren 2402.c dosyası programa eklenmiştir. Sonsuz döngüde do-while döngüsü kullanılmıştır. Bu döngüde klavye ile girilen bilgi RS232 port'undan alınmakta ve test edilmektedir. Döngüdeki `while ( (islem!='O') && (islem!='Y') );` komutu ile klavye ile girilen bilgi "O" ve "Y" dışında ise bu döngünün sürekli olarak dönmesi sağlanmaktadır. Yine aynı do-while döngüsünde `islem=toupper(islem);` komutu ile RS232 port'undan gelen bilgi `toupper()` fonksiyonu ile büyük karaktere çevrilmektedir. Yani gelen karakter "y" ise bu fonksiyon ile "Y" şeklinde büyük karaktere çevrilecektir. Programda 2402.c dosyasında bulunan önceden tanıtılan `read_ext_eeprom(adres)` komutu ile harici EEPROM'dan bilgi okunmakta, `write_ext_eeprom(adres,veri);` komutu ile de harici EEPROM'a veri yazılmaktadır.

19.1.7. Kendi Harici EEPROM Fonksiyonlarımızı Yazma

CCS C derleyicisi I^2C seri iletişimi için ana fonksiyonları içermektedir. Ayrıca CCS C kütüphaneleri içinde birçok harici EEPROM için hazır kontrol fonksiyonları bulunmaktadır. Bir önceki örnekte de hazır olarak var olan 2402.c dosyası kullanılmıştır. Fakat bu hazır dosyaları kullanmak yerine kendimiz de istediğimiz entegre için fonksiyonlar yazabiliriz. Aşağıda 24C02 EEPROM'u için yazılmış fonksiyonlar verilmiştir. Komutların yanında ilgili açıklamalar belirtilmiştir. Bölüm 19.1.4. gerekirse tekrar incelenmelidir. Bu fonksiyonlar **kendi_2402.c** adında kaydedilmiştir. Böylece kendimiz de kendi entegremiz için fonksiyon dosyası oluşturmuş olmaktayız.

```
//************************* EEPROM YAZMA FONKSİYONU *****************
void eeprom_yaz(byte adres, byte veri)
{
  i2c_start();          // i2c iletişimi başlangıç ayarları
  i2c_write(0xa0);      // EEPROM entegresine kontrol bilgisi gönderiliyor
  i2c_write(adres);     // adres bilgisi EEPROM'a gönderiliyor
  i2c_write(veri);      // veri bilgisi EEPROM'a gönderiliyor
  i2c_stop();           // i2c iletişimi sonlandırılıyor
  delay_ms(11);         // Belli bir süre bekleniyor
}

//************************* EEPROM OKUMA FONKSİYONU *****************
byte eeprom_oku(byte adres)
{
  byte veri;

  i2c_start();          // i2c iletişimi başlangıç ayarları
  i2c_write(0xa0);      // EEPROM entegresine kontrol bilgisi gönderiliyor
  i2c_write(adres);     // adres bilgisi EEPROM'a gönderiliyor
  i2c_start();          // i2c iletişimi başlangıç ayarları
  i2c_write(0xa1);      // EEPROM entegresine kontrol bilgisi gönderiliyor
  veri=i2c_read(0);     // Slave konumunda olan EEPROM'dan okuma yap
  i2c_stop();           // i2c iletişimi sonlandırılıyor
  return(veri);         // Fonksiyon "veri" değeri ile geri döner
}
```

Harici EEPROM adres bilgileri ve bu adreslere yazılacak verilerin heksadesimal olması gereklidir. Bunun için bir önceki uygulamada CCS C'de hazır bulunan "input.c" dosyası kullanılmıştı. Bu iş için bizde kendi fonksiyonlarımızı yazabiliriz. Aşağıda bizim tarafından yazılan, RS232 port'undan alınan verileri heksadesimal olarak alan fonksiyon verilmiştir. Bu fonksiyonları da **kendi_input.c** adında kaydederek istediğimiz programlarda kullanabiliriz.

```
//******** HEX OLARAK 1 BASAMAKLI BİLGİYİ OKUMA FONKSİYONU ********
byte hex_oku1 () // Klavyeden girilen 1 basamaklı bir değeri HEX formatında okur
{
  char karakter; // Karakter tipinde değişken tanımlanıyor

  karakter=getch(); // RS232 port'undan gelen bilgi alınıyor ve "karakter" değişkenine
                    // aktarılıyor
  putc(karakter); // Alınan karakter ekranda görünmesi için gönderiliyor

  if (karakter<='9') // alınan karakter eğer 9'dan küçükse
    return(karakter-'0');
  else // alınan karakter eğer 9'dan büyükse
    return(toupper(karakter)-'A')+10; // Dönüştürülen değer ile fonksiyon geri dönsün
}

//********* HEX OLARAK 2 BASAMAKLI BİLGİYİ OKUMA FONKSİYONU ***
byte hex_oku2() // Klavyeden girilen 2 basamaklı bir değeri HEX formatında okur
{
  int high,low; // tam sayı tipinde değişkenler tanımlanıyor

  high=hex_oku1(); // Alınan bilginin yüksek değerlikli 4 bit'ini al ve "high" değişkenine
                   // kaydet
  low=hex_oku1(); // Alınan bilginin düşük değerlikli 4 bit'ini al ve "low" değişkenine
                  // kaydet
  return(high*16+low); // Dönüştürülen değer ile fonksiyon geri dönsün
}
```

Hex_oku1 fonksiyonunda klavyeden girilen bilgi karakter tipindeki "karakter" değişkenine karakter=getch() komutu ile kaydedilir. Daha sonra klavyeden girilen karakterin ekranda görülmesi için putchar(karakter) komutu ile bu değer ekrana gönderilir. Sonra bu değerin ASCII olarak karşılığı 9 rakamının ASCII olarak karşılığından küçük ve/veya eşit olup olmadığına göre değerlendirilmektedir. Bu değerlendirmeye göre bir formülle hesaplanan değer fonksiyonun geri dönüş değeri olmaktadır. Hex_oku2() fonksiyonunda ise klavyeden girilen 2 basamaklı bilginin yüksek değerlikli kısmı ve düşük değerlikli kısmı hex_oku1 () fonksiyonu ile alınmaktadır. Daha sonra belirlenen formülle fonksiyonun geri dönüş değeri hesaplanmaktadır.

Örneğin klavyeden 5A değerinin girilmiş olduğunu farz edelim. Hex_oku2() fonksiyonu bu değerin 5 bilgisini **high**, A bilgisini **low** değişkenine hex_oku1 fonksiyonu aracılığı ile aktarır. `high=hex_oku1();` komutu kullanıldığında, hex_oku1 fonksiyonunda 5 değeri karakter değişkenine aktarılır. 5 değeri 9'dan küçük olduğundan fonksiyonun dönüş değeri belirlenen formüle göre hesaplanır. Yalnız alınan karakter değerleri derleyici tarafından ASCII koda çevrildiğinden formülde karakterlerin ASCII kod karşılığı yazılmalıdır.

Karakter-'0' = 35 (5'in ASCII karşılığı) - 30 (0'ın ASCII karşılığı) = 5

low=hex_oku1(); komutu kullanıldığında ise hex_oku1 fonksiyonu ile A değeri karakter değişkenine aktarılır. A değeri 9'dan büyük olduğu için fonksiyonda verilen formüle göre hesaplama yaparsak;

toupper(karakter)-'A')+10=(A-A)+10=41(A'nın ASCII karşılığı)-41(ASCII A)+10=10

bulunur. 10 yani ASCII olarak "A" değeri hesaplanır.

Hex_oku2 ( ) fonksiyonda verilen formüle bu değerleri yazarsak;

high*16+low = 5*16+10 = 80+10 = 90 değeri çıkar.

Bu değerin ASCII tablosunda karşılığı ise 5A'dır. Böylece klavyeden girilen 2 basamaklı sayının heksadesimal karşılığı okunmuş olur.

19.1.8. Kendi Harici EEPROM Fonksiyonlarımız İle 24C02 EEPROM Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☺

Bu uygulamada kendi oluşturduğumuz **kendi_input.c** ve **kendi_input.c** dosyalarını kullanarak Bölüm 19.1.5'de ki uygulamamız gerçekleştirilmiştir. Bu uygulama içinde Şekil-19.8'deki devre şemasının aynısı kullanılmıştır.

- Tanımlanan **"kendi_2402.c"** ve **"kendi_input.c"** dosyalarımızı kendi programlarınızda kullanmak için istenen bir isimde herhangi bir editör programı veya CCS C ile (File menüsünden "New" kısmına, oradan "Source File" komutu ile) uzantısı ".C" olacak şekilde kaydettikten sonra **C:\Program Files\PICC\Drivers\ dizini içine kopyalamanız gerekmektedir.** Aynı şekilde eğer dosyanızı farklı dizinde (klasörde) tutmak istiyorsanız ve bu şekilde dosyayı programınıza derleme işlemi için eklemek istiyorsanız `#include <D:\Deneme\kendi_2402.c>` komutunu kullanarak dosyanızın olduğu dizini programa tanıtmanız gerekmektedir.

Devre programı;

```
/*****************************************************************
 PIC16F877 ile Kendi Harici EEPROM Fonksiyonlarımız İle 24C02
                    EEPROM Uygulaması
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları
```

```
#use delay(clock=4000000)     // Gecikme fonksiyonu için kullanılan
                              // osilatör frekansı belirtiliyor
#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)
// RS232 ayarları belirtiliyor

#use i2c(master,sda=pin_c4,scl=pin_c3,slow=100000)//I²C Fonksiyonu,
            // SDA ve SCL pin'leri ve iletişim hızı belirleniyor
#include <kendi_input.c> // kendi_input.c dosyası programa ekleniyor
#include <kendi_2402.c>  // kendi_2402.c dosyası programa ekleniyor

byte veri,adres; // byte tipinde değişken tanımlanıyor
char islem;      // char tipinde değişken tanımlanıyor

//******************* ANA PROGRAM FONKSİYONU *******************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   printf("\n\r*****************************************************");
   printf("\n\r  24C02C Harici EEPROM Bilgi Okuma Yazma  ");
   printf("\n\r*****************************************************");
   printf("\n\n\r Bir islem seciniz>");

   while(1) //Sonsuz döngü
   {
      do // do-while döngüsü tanımlanıyor
      {
         printf("\n\rOkuma (O) veya Yazma (Y) >");
         islem=getc();// RS232 port'undan gelen veri alınıyor
         islem=toupper(islem); // toupper fonksiyonu karakterleri
                               // büyük karaktere çevirir (a>A)
         putc(islem); //"islem" değişkeni içeriği RS232 port'una
                      // gönderiliyor
      } while ( (islem!='O') && (islem!='Y') ); // Girilen karakter
                        // "O" veya "Y" dışında ise döngü başına dön

      if (islem=='O') // Eğer okunan karakter 'O' ise
      {
         printf("\n\rOkumak istediğiniz adresi giriniz> ");
         adres=hex_oku2();// RS232 port'undan gelen veriyi
                          // heksadesimal formatında al
         printf("\n\rDeger= %X",eeprom_oku(adres) ); // "adres"
// değişkeninin tuttuğu adresteki bilgiyi oku ve RS232'ye gönder
      }
```

```
        if (islem=='Y') // Eğer okunan karakter "Y" ise
        {
           printf("\n\rBilgi yazmak istediğiniz adresi giriniz> ");
           adres=hex_oku2();// RS232 port'undan gelen veriyi
                            // heksadesimal formatında al
           printf("\n\rBilgiyi giriniz> ");
           veri=hex_oku2();// RS232 port'undan gelen veriyi
                            // heksadesimal formatında al
           eeprom_yaz(adres,veri); // "adres" değişkeninin belirttiği
                             // adrese "veri" değişkeni değerini yaz
        }
    }
}
/*****************************************************************
```

19.2. SPI SERİ İLETİŞİM

SPI (**S**erial **P**eripheral **I**nterface) seri iletişimi I^2C iletişimi gibi az sayıda pin ile sağlanan bir seri iletişim metodudur. SPI iletişimde veriler 8 bit uzunluğunda senkronize olarak alınır ve gönderilir. Bazı PIC denetleyicileri içinde bulunan MSSP (The **M**aster **S**ynchronous **S**erial **P**ort) modülü SPI iletişimini sağlayacak donanımı içermektedir. SPI iletişimde toplam 3 adet pin kullanılır. Eğer slave modunda iletişim yapılacaksa bir uç daha kullanılır. Bunlar;

SDO - Serial Data Out - Seri Data Çıkış Pin'i, RC5/SDO pin'i.
SDI - Serial Data In - Seri Data Giriş Pin'i, RC4/SDI/SDA pin'i.
SCK - Serial Clock - Seri Saat Sinyali Pin'i, RC3/SCK/SCL .

Slave modunda iletişim yapılacaksa,
$\overline{SS}$ - Slave Select Pin'i, RA5/AN4/$\overline{SS}$

CCS C derleyicisi içinde SPI iletişimi kolayca kontrol etmemizi sağlayan fonksiyonlar bulunmaktadır. SPI İletişimi ile ilgili daha ayrıntılı bilgiyi kullandığınız PIC denetleyicinin datasheet dosyasında MSSP modülü başlığı altında bulabilirsiniz.

19.2.1. SETUP_SPI(), SPI_READ(), SPI_WRITE(), SPI_DATA_IS_IN() Fonksiyonları

Bu fonksiyonlar ile CCS C'de kolayca SPI iletişimi sağlanabilinmektedir.

SETUP_SPI( ) Fonksiyonu
Seri port arayüzünün (SPI) kurulum ayarlarını yapmaya yarayan fonksiyondur. Fonksiyondaki *"mod"* kısmına aşağıda verilen mod sabitleri yazılabilir.

setup_spi(*mod*);

Fonksiyonda *"mod"* kısmına aşağıda verilen sabit tanımlamalar yazılabilir.

SPI_SS_DISABLED = SPI kapalı
SPI_MASTER = SPI Master modunda, SCK clock sinyali dışarıya verilir.
SPI_SLAVE = SPI Slave modunda, SCK clock sinyali dışarıdan alınır.
SPI_L_TO_H = SCK sinyalinin yükselen kenarında veri gönderilir.
SPI_H_TO_L = SCK sinyalinin düşen kenarında veri gönderilir.

Aşağıdaki tanımlamalar master mod seçeneğinde kullanılır.
SPI_CLK_DIV_4 = SCK sinyalini 4'e böler.
SPI_CLK_DIV_16 = SCK sinyalini 16'ya böler.
SPI_CLK_DIV_64 = SCK sinyalini 64'e böler.
SPI_CLK_T2 = SCK sinyali olarak Timer2/2 çıkış frekansı kullanılır.

Örnek;

```
setup_spi(SPI_MASTER | SPI_H_TO_L | SPI_CLK_DIV_16);
```

SPI_READ( ) Fonksiyonu
SPI biriminden bilgi okumaya yarayan fonksiyondur. Eğer girişte bilgi (data) varsa o bilgiyi okur ve o değerle fonksiyon geri döner, eğer girişte bilgi yoksa bilgi gelene kadar bekler. Fonksiyondaki *"data"* kısmı 8 bit'liktir.

***değişken ismi* = spi_read( );**

SPI_WRITE( ) Fonksiyonu
SPI biriminden bir byte gönderme işlemini yapar. Fonksiyondaki *"değer"* kısmı 8 bit'liktir.

spi_write(*değer*);

SPI_DATA_IS_IN( ) Fonksiyonu
SPI biriminden herhangi bir bilgi (data) alındığında fonksiyon 1 (TRUE) değeri ile geri döner. SPI biriminden herhangi bir bilgi alınmadığında 0 (FALSE) değeri ile geri döner.

***değişken ismi* = spi_data_in( );**

Yukarıda tanımlanan fonksiyonlar ile SPI iletişimine sahip herhangi bir eleman ile çok kolayca iletişim kurulabilir.

20. Bölüm

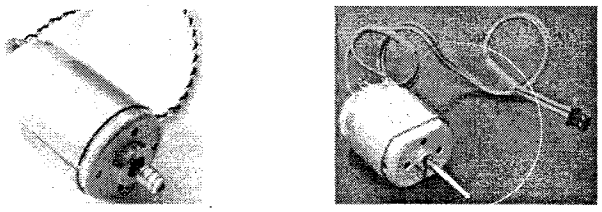
MOTOR UYGULAMALARI

- DC Motor ve Uygulaması (L298 Entegresi İle)
- Step (Adım) Motor ve Uygulaması (ULN2003 İle)
- Bilgisayardan Step Motor Kontrol Uygulaması
- R/C Servo Motor ve Uygulaması

Bu bölümde DC motor, step (adım) motor ve R/C tipi servo motorlar hakkında bilgi verilmiş ve ardından her tip motor için birer uygulama gerçekleştirilmiştir.

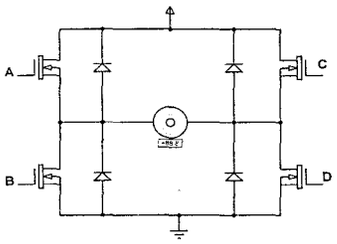
20.1. DC MOTOR

DC motorlar endüstride uzun zamandan beri çok kullanılan motor tipidir. DC motorlar manyetik alan içine içinden akım geçen telin konulmasıyla meydana gelen kuvveti kullanma prensibiyle dönmeyi sağlarlar. Diğer elektrik motor türleri de bu temel prensibi değişik donanım özellikleriyle kullanır. DC motorlar, sabit bir mıknatıs ve içinde bir rotor (pervane) içerirler.



Şekil-20.1. *DC motorlar.*

DC motorları direkt olarak mikrodenetleyici çıkışı ile süremeyiz. Bu nedenle sürücü devreler kullanılır. DC motorların sürücü ve yön kontrolü için H-köprü devreleri kullanılır. Köprü devreleri genelde transistör ile yapılır. Fakat hem yön hem de hız kontrolü için köprü devrelerinde transistör yerine MOSFET kullanılabilir. Şekil-20.2'de verilen MOSFET'li, bir H-Köprü devresi ile kontrol uçlarına Tablo-20.1'de verilen lojik değerler uygulandığında istenen ileri, geri, dur, serbest konumları elde edilir. Yine A ve D veya B ve C MOSFET çiftine uygulanan **gate sinyali** PWM mantığına göre kontrol edilirse motorun hızı da kontrol edilmiş olur.



A	B	C	D	Motor Durumu
1	0	0	1	Saat yönünde döner
0	1	1	0	Saat yönünün tersine döner
1	0	1	0	Motor durur (OFF)
0	0	0	0	Motor boşta

Şekil-20.2. *MOSFET'li H-Köprü sürücü devresi.*

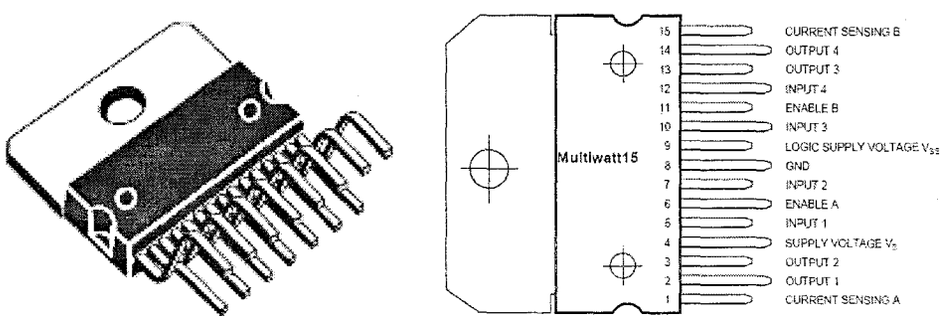
Tablo-20.1. *MOSFET'li H köprü kontrol durumları.*

Şekil-20.2'deki diyotlar, DC motor tarafından oluşabilecek ters EMK'nın (Elektro Motor Kuvveti) devre elemanlarına zarar vermesini önlemek için kullanılmıştır. Bu iş için BA159 hızlı diyotu gibi diyot modelleri tercih edilir. Tüm MOSFET'ler TTL/CMOS çıkışları ile sürülemez. Sadece bazı MOSFET'ler TTL/CMOS uyumludur. Bu nedenle ya TTL/CMOS uyumlu MOSFET kullanmak gerekli ya da MOSFET sürücü entegreleri kullanmak gereklidir. MOSFET sürücü entegreler olarak MAX620/621, ICL7667, MAX8552 gibi sürücü entegreler kullanılabilir.

Fakat çoğu DC motor kontrol uygulamasında Şekil-20.2'de verilen H-köprü sürücü devrelerini içinde barındıran entegre elemanlar kullanılır. L293, L298, L6201, L6202, L6203 veya LMD18200 gibi entegreler DC motor sürücü için kullanılabilir. Biz bu bölümde DC motor kontrolü için L298 entegresini inceleyeceğiz.

20.1.1. L298 DC Motor Sürücü Entegresi

L298 DC motor sürücü entegresi, içinde iki adet transistörlü sürücü devre barındırır. Sürücü devrelerin ilki A, ikincisi B diye isimlendirilmiştir. Böylece aynı entegre ile 2 adet motor kontrol edilebilir. Çalışma gerilimi 46V'a kadardır. Çıkış akımı 4 Amper'e kadar çıkmaktadır. Entegre aynı zamanda aşırı ısı korumasına sahiptir.



Şekil-20.3. *L298 Multiwatt15 kılıfı ve pin isimleri.*

Pin No	Pin İsmi	Açıklama
1,15	Sense A, Sense B	Bu pin'ler şaseye bir direnç ile bağlanır. Motordan geçen akımı kontrol etmek için kullanılır.
2,3	Out1, Out2	A köprü devresinin çıkış uçları.
4	Vs	Motor beslemesinin + ucu. Bu uç 100nF'lık kondansatör ile şase ucuna bağlanmalıdır.
5,7	Input1, Input2	A köprü devresinin TTL kontrol uçları.
6,11	Enable A, Enable B	Entegre içindeki köprü devrelerini (A ve B) aktif etme pin'leri.
8	GND	Şase ucu.
9	VSS	L298 entegresi besleme gerilim ucu. Bu uç 100nF'lık kondansatör ile şase ucuna bağlanmalıdır.
10,12	Input3, Input4	B köprü devresinin TTL kontrol uçları.
13,14	Out3, Out4	B köprü devresinin çıkış uçları.

Tablo-20.2. *L298 Entegresi Multiwatt15 kılıfı pin açıklamaları.*

L298 entegresinden Input girişlerine verilen lojik değerler (0,1) ile motorun ileri, geri, stop veya boşta olma durumları belirlenir. Bu durumlar Tablo-20.3'de verilmiştir.

Köprü Devresi	Girişler	Durum	Köprü Devresi	Girişler	Durum
A Köprü Devresi	Enable A=High Enable B=Low Input1=High Input2=Low	İLERİ	B Köprü Devresi	Enable A=High Enable B=Low Input1=High Input2=Low	İLERİ
	Enable A=High Enable B=Low Input1=Low Input2=High	GERİ		Enable A=High Enable B=Low Input1=Low Input2=High	GERİ
	Enable A=High Enable B=Low Input1=Input2	HIZLI STOP		Enable A=High Enable B=Low Input1=Input2	HIZLI STOP
	Enable A=Low Enable B=Low Input1=X Input2=X	MOTOR BOŞTA		Enable A=Low Enable B=Low Input1=X Input2=X	MOTOR BOŞTA

Tablo-20.3. *L298 girişlere göre çalışma durumları.*

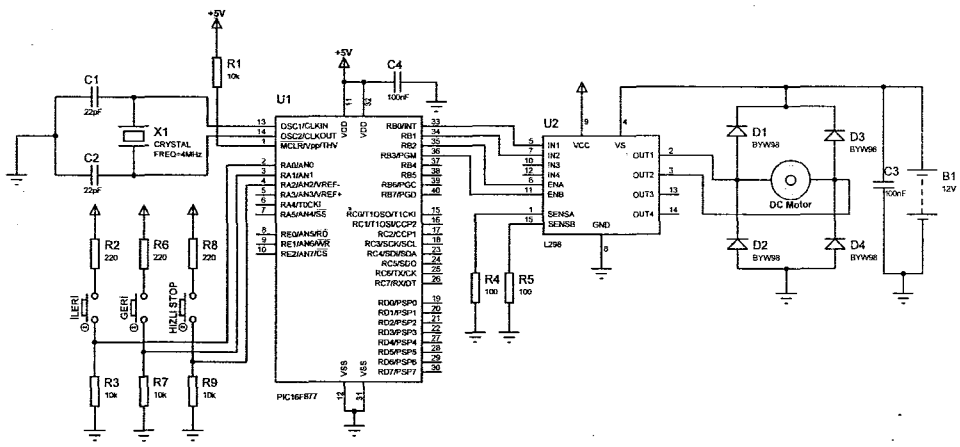
20.1.2. DC MOTOR UYGULAMASI

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada 12V'luk bir DC motorun yön kontrolü L298 entegresi kullanılarak gerçekleştirilmiştir. Uygulamada L298 entegresinin Köprü-A kısmı kullanılmıştır. A port'una bağlanan 3 buton ileri, geri ve hızlı durdurma işlemleri içindir. L298'in 1

numaralı ucuna bağlanan direnç değeri küçültülerek motordan geçen akım arttırılabilir, direnç değeri arttırılarak motordan geçen akım azaltılabilir. Devrede kullanılan diyotlar hızlı diyotlardır. Kendi uygulamanızda kullandığınız motor voltajı ve motorun çekeceği akıma göre diyotlar kullanmayı unutmayınız. Kullandığınız motorun + besleme voltajı için Vs ucuna uygun gerilim vermeyi unutmayınız. **Eğer uygulamanızda motorun hızını da ayarlamak istiyorsanız LMD18200 gibi PWM girişli entegreleri kullanabilirsiniz. PWM sinyali için denetleyicide bulunan CCP modülünde PWM birimini kullanabilirsiniz.**

Devrenizin çalışmasını PIC PROG/DEKA deneme kartında görmek için B Port'u çıkış terminalinden tel ile ilgili çıkışları bir board üzerine kurduğunuz L298 ve DC motor bağlantılarına verebilirsiniz.



Şekil-20.4. *DC motor uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile DC Motor Uygulaması
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.

// Giriş ve çıkış pin'lerine isim atanıyor
#define buton_ileri pin_a0
#define buton_geri  pin_a1
#define buton_stop  pin_a2
#define input1      pin_b0
#define input2      pin_b1
#define enable_a    pin_b2
#define enable_b    pin_b3
```

```
/********* ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   output_high(enable_a); // Köprü A seçili
   output_low(enable_b);  // Köprü B pasif

   while(1) // Sonsuz döngü
   {
      if (input(buton_ileri)) // İLERİ Butonuna basılsı ise
      {                       // Motor İLERİ yönde dönsün
         output_high(input1); // L298 Input1 girişi lojik-1
         output_low(input2);  // L298 Input2 girişi lojik-0
      }

      if (input(buton_geri))  // GERİ Butonuna basılsı ise
      {                       // Motor GERİ yönde dönsün
         output_low(input1);  // L298 Input1 girişi lojik-0
         output_high(input2); // L298 Input1 girişi lojik-1
      }
      if (input(buton_stop)) // HIZLI STOP Butonuna basılsı ise
      {                      // Motor HIZLI STOP olarak dursun
         output_low(input1); // L298 Input1 girişi lojik-0
         output_low(input2);  //  L298  Input1  girişi  lojik-0
// Hızlı stop için burada her iki girişte lojik-1 olabilirdi.
//Önemli olan Hızlı Stop için, her iki girişin aynı seviyede olması
      }
   }
}
//*****************************************************************
```

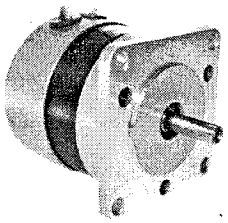
Programda ilk başta denetleyicinin giriş ve çıkış olarak kullanılan pin'lerine isim atanmıştır. Böylece programda anlaşılabilirlik arttırılmıştır. Sonsuz döngüden önceki `output_high(enable_a);` komutu ile Enable A ucu lojik-1 yapılarak L298 entegresinde A köprüsünün kullanılacağı belirtilmiştir. `output_low(enable_b);` komutu ile de Enable B ucu lojik-0 yapılarak L298 entegresinde B köprüsü pasif edilmiştir.

Sonsuz döngüde `if (input(buton_ileri))` komutu ile RA0 pin'ine bağlı "İLERİ" butonuna basılıp basılmadığı kontrol edilmiştir. Bu butona basılmışsa `output_high(input1);` komutu ile Input1 girişi lojik-1, `output_low(input2);` komutu ile Input2 girişi lojik-0 yapılmıştır. Böylece L298 motoru İLERİ yönde hareket ettirecek şekilde ayarlanmış olmaktadır. Diğer if döngüleri ile de aynı

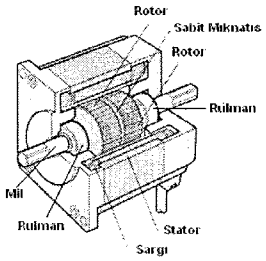
mantıkla diğer butonlara basılıp basılmadığı kontrol edilmiş ve istenen çalışma durumu için girişlere Tablo-20.3'deki değerler uygulanmıştır.

20.2. STEP (ADIM) MOTOR

Step (adım) motorlar çok yönlü, kalıcı ömürlü birçok uygulamada kullanılan motorlardır. Step motorlarda diğer motor türleri gibi elektrik enerjisini, mekanik enerjiye çevirirler. Diğer türlerden farkı, dijital kontrollü olması ve her gönderilen pals için belli açıda dönmesidir. Bu özelliğinden dolayı bilgisayar destekli uygulamalarda sıkça kullanılmaktadır. Temelde, sabit mıknatıs içeren rotor (hareketli kısım) ve rotorun hareketini sağlayan statordan (sabit kısım) oluşmaktadır. Step motorlar robotik alanında, printer, floppy disk, hard disk gibi bilgisayar donanımında, kamera, konum kontrollü sistemlerde ve tıp elektroniğinde yaygınca kullanılmaktadır.

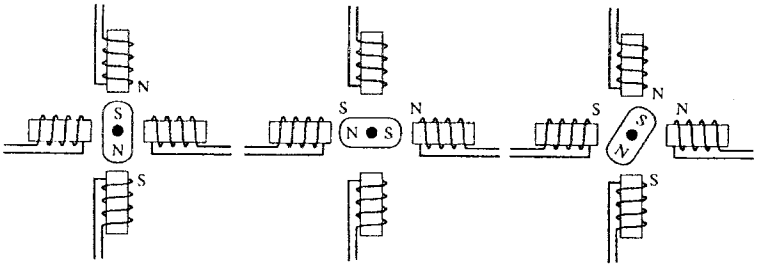


Şekil-20.5. *Step motor.*



Şekil-20.6. *Step motor iç yapısı.*

Step motorlar sabit mıknatıslı ve karma mıknatıslı yapılara sahiptir. Sabit mıknatıslı step motorlar mıknatısın manyetik alan içinde hareket etmesi prensibine göre çalışır. Sargılara uygulanan sinyallere göre rotor hareket eder. Şekil 20.7'de adım motorun basitleştirilmiş çalışma prensibi görülmektedir. Şekil-20.7'de görülen stator sargılarından (bobin) gerçek step motorlarında çok miktarda bulunmaktadır. Sargılara uygulanan sinyallerden dolayı sargılar içinden geçen akımın yönüne göre sargılarda N veya S kutbu oluşur. Rotor sargısı da bu duruma göre döner.



Şekil-20.7. *Step motor çalışma prensibi.*

Step motorlar genellikle kare dalga palsler ile enerjilendirilir. Rotorda oluşan tork, bobinlere (sargılar) uygulanan akımla doğru orantılıdır. Uygulanan akım ne kadar

fazla ise, oluşan manyetik alan o kadar fazla olur, dolayısıyla tork da artar. Bobin sargıları doyuma ulaşınca akım ve gerilim arttırılsa dahi tork artmaz.

Karma step motorlarda (Hybrid stepper motors) rotor, iki tane aynı doğrultuda olan silindirin paslanmaz çelik şafta (mil) monte edilmesi ile oluşur. Böylece rotorun bir ucu kuzey (N) diğer ucu güney polaritesini (S) belirtir. Rotor silindiri dişlere sahiptir. Bu tip motorlarda rotor diş sayısı ile sargı kutup sayısı birbirinden farklı olabilir. Bu motora daha küçük açılarda hareket yapma imkânı tanır.

Adım motorlar genelde 4, 5 ve 6 uçlu olarak piyasada bulunmaktadırlar. En fazla olarak 5 uçlu adım motorlar kullanılmaktadır. **5 uçlu (kabloya) sahip adım motorlarda bir uç ortak uçtur, 6 uçlu adım motorlarda ise 2 ortak uç vardır.** Ortak uçlar ya + beslemeye, ya da toprak hattına bağlanır. Sonuçta kalan 4 uç bobin-sargı (stator) uçlarıdır. Bobin uçlarına + besleme veya toprak seviyesi uygulanır. Bobin uçlarını bulmak için sargı uçlarının direnci ölçülmelidir. Daha sonra da deneme yöntemi ile doğru sargı uçları bulunabilir.

Motora bir pals uygulandığında, rotorun hareket edebileceği açı miktarına **adım açısı (step angle)** denir. Piyasadan step (adım) motor alırken adım açısına göre alınır. Örneğin $1{,}8^0$ adım açısına sahip adım motor 360^0 için 200 adım hareket ettirilir. Adım açısı ne kadar küçük olursa motor o kadar hassas kullanabilir. Step motora sinyal uygulandığında belli bir zaman sonra tepki verir. Bu zaman miktarına **adım cevabı (adım response)** denir. Motorun 1 saniyede yapabileceği maksimum adım sayısına ise **adımlama oranı (stepping rate)** denir. Step motoru sürmek için her bobine sırasıyla enerji verilerek dönme hareketi sağlanabilir **(Tek faz uyartım yöntemi)**. Fakat bu yöntemden ziyade genelde **İki faz uyartım yöntemi** kullanılır. İki faz uyartım yöntemi için 2 farklı sürme biçimi vardır. Bunlar **"Tam Adım"** ve **"Yarım Adım"** sürme tekniklerdir. **Tam adım** yönteminde motor sargılarının ikisi aynı anda enerjilendirilir. Bu sayede tak faz uyartım yöntemine göre yaklaşık 1,5 kat daha fazla tork elde edilir. Fakat çekilen akımda yaklaşık 2 katına çıkmaktadır. **Yarım adım** yönteminde ise tek faz ve tam adım sürme adımları ard arda uygulanır ve böylece rotor her enerjiendirmede yarım adım döner. Bu sayede 3^0'lik dönme açısına sahip bir step motoru, yarım adım sürme yöntemiyle $1{,}5^0$'lik dönme açısına sahip olarak döndürebiliriz. Yarım adım sürmede motor daha hassas açılarla hareket ettirilir.

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

Tablo-20.4. *Tam adım sürme.*

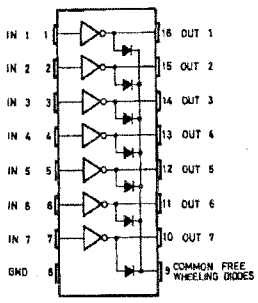
Adım	Uç-1	Uç-2	Uç-3	Uç-4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Tablo-20.5. *Yarım adım sürme.*

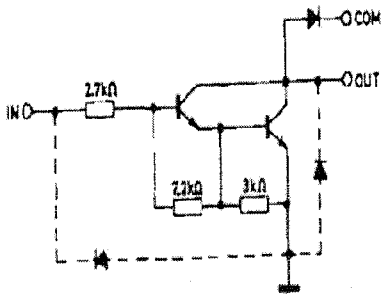
Adım motor sürme işlemlerinde PIC denetleyici tarafından lojik-1 veya lojik-0 değerleri bobinlere uygulanır. Fakat PIC denetleyiciler en fazla 25 mA akım verebilirler. Bu akım değeri adım motoru sürmek için yeterli değildir. Bu sebeple denetleyici ucundan gelen sinyal akım yükseltme işlemine tabi tutulmalıdır. Bu işlem için transistörlerden faydalanabilir. Fakat bunun yerine tümleşik entegrelerde kullanılabilir. ULN 2003 entegresi step motor sürücü entegresi olarak yaygın şekilde kullanılmaktadır.

20.2.1. ULN2003 Entegresi

ULN2003 entegresi içinde 7 adet NPN tipi darlington transistör çifti ve her transistör çifti çıkışında zıt EMK'yı (Elektro Motor Kuvveti) önlemek için bir diyot barındırır. Şekil-20.8'de ULN2003A entegresinin pin diyagramı ve Şekil-20.9'da da içerdiği transistör çiftlerinden birisinin bağlantısı görülmektedir. Bu entegre sayesinde çıkışlar 500mA'lik akım verebilmektedir. Bu entegreler ile (ULN2002A, ULN2003A, ULN2004A) 50V'a kadar çıkış sağlanabilir. ULN2003 girişine lojik-1 uygulandığında aynı hat çıkışından lojik-0 alınır.



Şekil-20.8. *ULN2003A pin diyagramı.*



Şekil-20.9. *ULN2003A transistör çifti.*

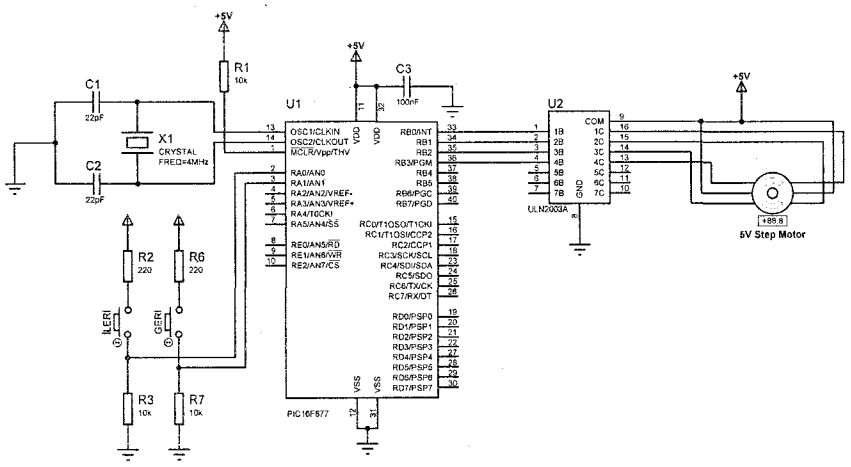
20.2.2. Step Motor Uygulaması (ULN2003A İle)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada 5V'luk bir step motor, ULN2003A entegresi ile sürülmüştür. Devrede RA0 ve RA1 girişlerine İLERİ ve GERİ butonları bağlanmıştır. Butonlara bir kez basıldığında step motor sadece 1 adım ileri veya geri gider. Program step motorun yarım adım metoduna göre adım atacak şekilde yazılmıştır. Siz kendiniz isterseniz tam adım yöntemine göre de motorunuzun adımlarını ilerletebilirsiniz.

Proteus programında benzetim (simülasyon) yapmadan önce, step motorun üstüne fare ile ilk önce sağ, sonra sol tuş ile tıkladığınızda açılan "Edit Component" menüsünde "Nominal Voltage" kısmına step motorunuzun çalışma gerilimini girmeyi unutmayınız (Devredeki bağlantı 5V'luk motor içindir). Aynı menüdeki "Step Angle" kısmında ise motorunuzun adım açısı değerini girmeyi unutmayınız.

Devrenizin çalışmasını PIC PROG/DEKA deneme kartında görmek için B Port'u çıkış terminalinden tel ile ilgili çıkışları bir board üzerine kurduğunuz ULN2003 ve step motor bağlantısına verebilirsiniz. Sadece devre kartınız ile de devrenin çalışmasını B port'una bağlı LED'lerde görebilirsiniz. Program ile B port'una gönderilen adım değerleri deneme kartınızın LED'lerinde görülecektir.



Şekil-20.10. *Step motor uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 ile Step Motor Uygulaması (ULN2003A İle)
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
// Giriş ve çıkış pin'lerine isim atanıyor
#define buton_ileri pin_a0
#define buton_geri  pin_a1

int i=0,hiz=400;  // Tam sayı değişkenler tanımlanıyor

// Step motor yarım adım dönüş adımları
const int yarim_adim[]={0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09};

/******** ANA PROGRAM FONKSİYONU********/
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
```

```
   setup_timer_2(T2_DISABLED,0,1); // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);    // ANALOG giriş yok
   setup_adc(ADC_OFF);             // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);            // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);            // CCP2 birimi devre dışı

   output_b(0x00); // İlk anda B port'u çıkışı sıfırlanıyor
   while(1) // Sonsuz döngü
   {
      if (input(buton_ileri)) // İLERİ Butonuna basıldı ise
      {
         output_b(yarim_adim[i]); // Step motor 1 adım ileri
         while(input(buton_ileri));// Tuş bırakılana kadar bekle
         delay_ms(hiz);           // Adımlar arası bekleme süresi

         if (i==7)          // i değişkeni 7 olunca i değeri -1 olsun
            i=-1;
         i++;               // i değişkeni değerini 1 arttır
      }

      if (input(buton_geri)) // GERİ Butonuna basıldı ise
      {
         if (i==0)     // i değişkeni 0 ise i değeri 8 olsun
            i=8;
         i--;          // i değişkeni değerini 1 azalt
         output_b(yarim_adim[i]); // Step motora 1 adım geri
         while(input(buton_geri)); // Tuş bırakılana kadar bekle
         delay_ms(hiz);           // Adımlar arası bekleme süresi
      }
   }
}
//*****************************************************************
```

Programda `const int yarim_adim[]={0x01, 0x03, 0x02, 0x06, 0x04, 0x0C, 0x08, 0x09};` komutu ile step motor bobin uçlarına gönderilecek bobin değerleri yarim_adim[] dizisi ile belirtilmiştir. Sonsuz döngüde `if (input(buton_ileri))` komutu ile "İLERİ" butonuna basılıp basılmadığı kontrol edilmektedir. Eğer butona basılmışsa B port'una ilk adım değeri gönderilmektedir. `while(input(buton_ileri));` komutu butondan el çekilene kadar while döngüsünden çıkılmamasını sağlar. Bu sayede butona her basıldığında motor sadece 1 adım atar. `delay_ms(hiz);` komutu ile de "hiz" değişkeni ile belirtilen süre kadar (milisaniye olarak) adımlar arası bekleme verilmiştir. `if (i==7)` ve `i=-1;` komutları ile de i değeri 7 olmuşsa i değeri -1'e eşitlenmiştir. Bu komutların ardındaki `i++;` komutu ile de i değeri 0 olacak ve döngü tekrar 0. değerinden başlayacaktır. Her `i++;` komutu ile de i değeri her butona basılışta arttırılmaktadır.

`if (input(buton_geri))` komutu ile de "GERİ" butonuna basılıp basılmadığı kontrol edilmiştir. Bu döngüde de ilk döngüde ki mantık kullanılmıştır. Döngüde `i--;` komutu ile motora gönderilecek adım bilgileri her butona basılışta bir

azaltılarak motorun geriye dönmesi sağlanmıştır. Yine döngüdeki `if (i==0)` ve `i=8;` komutları ile i değeri 0 olduğunda i değerinin tekrar 8'den başlaması sağlanmıştır.

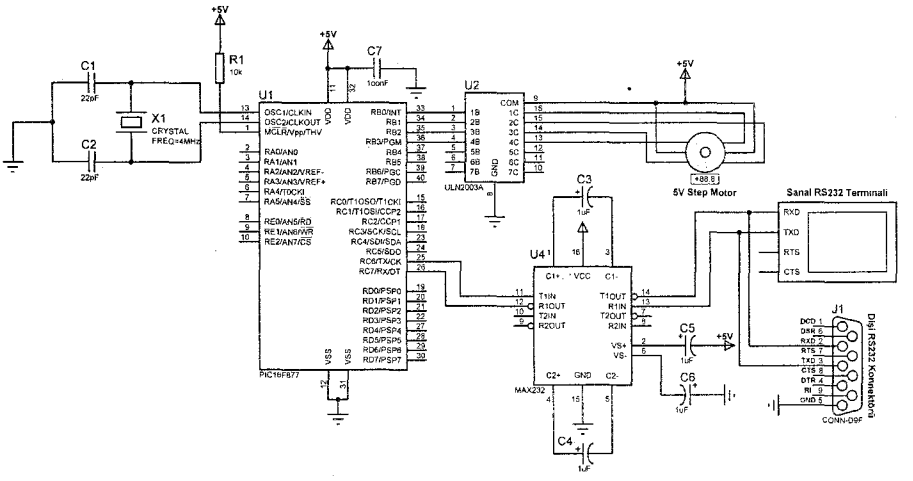
20.2.3. Bilgisayar İle Step Motor Kontrolü

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada bir bilgisayardan step motor kontrolü sağlanmıştır. Step motor sürücüsü için ULN2003A entegresi kullanılmıştır. Programda klavyeden girilen büyük R veya küçük r tuşu motorun sağa dönmesini, büyük L veya küçük l tuşu motorun sola dönmesini sağlamaktadır.

- Devreyi benzetim programında çalıştırdığımızda seri iletişimi görsel olarak görebilmek için Proteus-ISIS programında bulunan sanal RS232 Terminal (Virtual Terminal) devreye eklenmiştir. Böylece sanal olarak seri iletişim bağlantısı yapmış olmaktayız. Pratikte devreyi gerçekleştirirken bu sanal terminal devreye eklenmeyecektir. **Proteus programında sanal RS232 Terminali (Virtual Terminal) üzerine fare ile sağ tuş ardından sol tuş ile tıklanarak açılan kontrol menüsünde "Advanced Properties" kısmında "RX/TX Polarity" ayarının "Inverted" olarak seçildiğinden emin olunuz.**

- Devrenizin çalışmasını PIC PROG/DEKA deneme kartında görmek için B Port'u çıkış terminalinden zil teli ile ilgili çıkışları bir board üzerine kurduğunuz ULN2003 ve step motor bağlantısına verebilirsiniz. Sadece devre kartınız ile de devrenin çalışmasını B port'una bağlı LED'lerde görebilirsiniz. Program ile B port'una gönderilen adım değerleri deneme kartınızın LED'lerinde görülecektir.



Şekil-20.11. *Bilgisayar ile step motor kontrol uygulaması devre şeması.*

Gerçekte devrenizi denemek için Windows işletim sisteminde bulunan **HyperTerminal** programı (Başlat-Programlar-Donatılar-HyperTerminal) veya CCS C programında bulunan **siow.exe** programı (CCS C programında "Tools" menüsünde "Serial Port Monitor" komutu seçilerek veya C:\Program Files\PICC\siow.exe dizinine girilerek) çalıştırılabilir.

Devre programı;

```
/*******************************************************************
      PIC16F877 İle Bilgisayar İle Adım (Step) Motor Uygulaması
*******************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#use rs232 (baud=9600, xmit=pin_C6, rcv=pin_C7, parity=N, stop=1)
// RS232 protokolü ayarları belirtiliyor
int i,hiz=500; // Tam sayı tipinde değişkenler tanımlanıyor
char islem=0;  // Karakter tipinde değişken tanımlanıyor

// Step motor yarım adım dönüş adımları
const int yarim_adim[]={0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09};

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   output_b(0x00); // B port'u çıkışı başlangıçta sıfırlanıyor

/****************MENÜ*********************************/
   printf("\n\r*************************************************");
   printf("\n\r            ADIM MOTOR KONTROLÜ                  ");
   printf("\n\r*************************************************");
   printf("\n\n");
   printf("\n\rIslem Seciniz:\n");
   printf("\n\r-------------------------------------------------");
   printf("\n\r   Saga Dondur    (R)");
   printf("\n\r   Sola Dondur    (L)");
   printf("\n\r  Motoru Durdur(R ve L disindaki herhangi bir tus)");
   printf("\n\r-------------------------------------------------");
   printf("\n\rYapmak istediginiz islemi seciniz>");
```

```
   while(1)  // Sonsuz döngü
   {
      islem="null";// islem değişkeninin içeriğini boş olarak ayarla
      islem=getchar(); //RS232'den gelen karakter islem değişkeninde
      putc(islem); // Klavye'den girilen karakteri RS232'den gönder.

      while (islem=='R'||islem=='r') //Sağa döndürme seçildiyse
      {
         printf("\n\rMotor Saga Donuyor...");
         for (;i<8;i++)    // Sıra ile sağa dönme değerlerini gönder
         {
            output_b(yarim_adim[i]); // Step motor 1 adım ileri
            delay_ms(hiz);           // Adımlar arasındaki süre

            if (i==7)    // Eğer i değeri 7 olursa sıfırla
               i=0;

          if (kbhit()) break;//RS232'den veri alındıysa döngüden çık
         }
         if (kbhit()) break;//RS232'den veri alındıysa döngüden çık
}

      while (islem=='L'||islem=='l') //Sola döndürme seçildi ise
      {
         printf("\n\rMotor Sola Donuyor...");

         for (;i>=0;i--)    // Sıra ile sola dönme değerlerini gönder
         {
            if (i==0)    // Eğer i değeri 0 ise i değeri 7 olsun
               i=7;
            output_b(yarim_adim[i]);    // Step motor 1 adım geri
            delay_ms(hiz);              // Adımlar arasındaki süre

          if (kbhit()) break;//RS232'den veri alındıysa döngüden çık
         }
         if (kbhit()) break; //RS232'den veri alındıysa döngüden çık
       }
   }
}
/*****************************************************************
```

Programda önceki uygulamalara gördüğünüz komutlara göre bazı küçük yenilikler vardır. Öncelikle bu adım motor uygulamasında, önceki adım motor örneğinde olduğu gibi motor her butona basıldığında yarım adım sağa veya sola dönmemektedir. Sağa dönme tuşuna veya sola dönme işlemi seçildiğinde, klavyeden başka bir tuşa basılana kadar motor istenen yönde yarımşar adımlar halinde dönmektedir. Programda kullanılan **"null"** ifadesi "islem" değişkeninin değerini boşaltmak içindir. Yine programda kullanılan **"kbhit( )"** fonksiyonu, RX ucundan bir karakter alındığı zaman TRUE yani 1 değeri ile geri dönen bir fonksiyondur. Programda adımlar arasında verilen adım süresi "hiz" değişkeni değiştirilerek azaltılıp arttırılabilir.

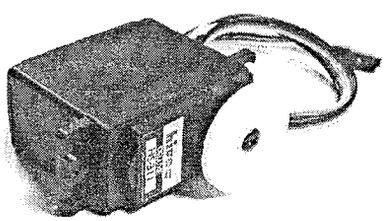
20.3. R/C SERVO MOTOR

Servo motorlarda DC motorların temelini kullanır. Fakat buna ek olarak başka bileşenleri de vardır. Servo motorun bileşenleri aşağıda verilmiştir.

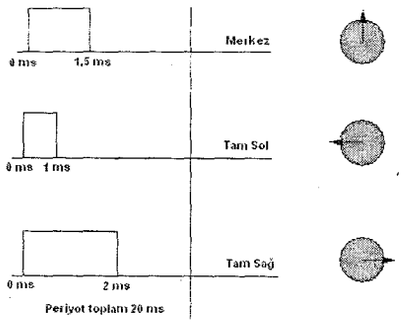
- DC motor
- Torku artırmak için dişli sistemi
- Elektronik şaft (mil) pozisyon ve kontrol devresi

Servo motorlar şaftın kaç derece döndüğünü ve hangi hızda döndüğünü algılar ve girişe geri besleme olarak verir. Böylece girişte istenen durum ile çıkış durumundan oluşan bir hata sinyali oluşur. Bu hata sinyaline göre servo motor şaftı döndürerek istenen pozisyon ve hızı sağlar. Bu algılama işlemi için servo motorlarda rotora takılı bir enkoder veya pozisyon algılayıcı potansiyometre bulunur. Bu tipteki servo motorlarda enkoder belirli sayıda boşluklardan oluşmaktadır. Mil döndükçe deliklerin sayısı pozisyon devresi tarafından sayılmaktadır. Servo motorun kaç tur atacağı dijital bilgi olarak servo motora bildirilir. Böylece pozisyon kontrol devresi kaç tur atacağını bilir ve o kadar tur atar.

R/C tipi servo motorlarda enkoder yoktur. Bunun yerine şafta bağlanmış ve dönüşü algılayan potansiyometre vardır. Bu tip servo motorlar **PWM** (**P**als **W**idth **M**odulation – Pals Genişlik Modülasyonu) tekniği ile çalışırlar. Kontrol ucuna gelen PWM sinyalinin görev çevrimine (duty cycle) göre belli açılarda dönme yaparlar. Bu tip servo motorlara R/C tip servo motor denmesinin nedeni genelde radyo frekansı ile uzaktan kontrol edilerek hobi amaçlı araba, uçak, helikopter vb. yapımlarında kullanılmasındandır. R/C kısaltması **R**adio **C**ontrolled kelimelerinin baş harflerinden meydana gelmiştir. R/C servo motorlar derece esasına göre dönerler. PWM sinyallerini yorumlayarak gerekli dönme derecelerini hesaplar ve o değer ölçüsünde dönüş yaparlar. Kontrol sinyali aynı kaldığı müddetçe konumlarını korurlar. R/C servo motorlar 0^0 ile 180^0 arasında dönme gerçekleştirirler. Bu dönme dereceleri PWM sinyalinin görev çevriminin genişliğine göre değişir.



Şekil-20.12. R/C servo motor.



Şekil-20.13. R/C servo motor kontrol sinyalleri

Dönmenin gerçekleşmesi için PWM sinyalinin frekansı 50 Hz olmalıdır. Yani R/C servo motorların kontrol sinyalinin periyodu 20 ms olmalıdır. 0^0 ile 180^0 arasındaki dönüş dereceleri ise PWM sinyalinin görev çevriminin(duty cycle)

yaklaşık **1 ms ile 2 ms** arasında değiştirilmesi suretiyle elde edilir. PWM sinyalinde 1 ms'lik görev çevrimi motoru merkez konuma (0^0) , 1,5 ms'lik görev çevrimi motoru tam sol (90^0 sola) konuma, 2 ms'lik görev çevrimi motoru tam sağ (90^0 sağ) konuma getirir. R/C servo motorların besleme gerilimleri ise DC 4,8 V ile 6 V arasında olmalıdır. R/C servo motorların üzerinde **tork güçleri** de belirtilmektedir. Bu tork gücü kg.cm cinsinden verilmektedir. Örneğin 6 volt besleme altında 3.7kg.cm. çıkış torkuna sahip bir R/C servo motor, motor miline 1 cm. uzaklıkta bağlanan 3.7kg. ağırlıktaki bir yükü kaldırabilir (Kuvvet = Yük ağırlığı x kol uzunluğu=3,7kg.x1cm=3.7kg.cm. - bu formülde yük ağırlığı kısmına kolun kendi ağırlığı da eklenmelidir-).

R/C servo motorun içyapısı, bir DC motor, bir potansiyometre, bir yükselteç devresi ve dişli çarklardan oluşmaktadır. Servo motora bir kontrol sinyali geldiğinde motor istenen pozisyon ile mevcut pozisyonunu karşılaştırır. Eğer istenen pozisyon mevcut pozisyondan küçük ise servo motor mili sola döner. Eğer istenen pozisyon mevcut pozisyondan büyük ise servo motor mili sağa döner.

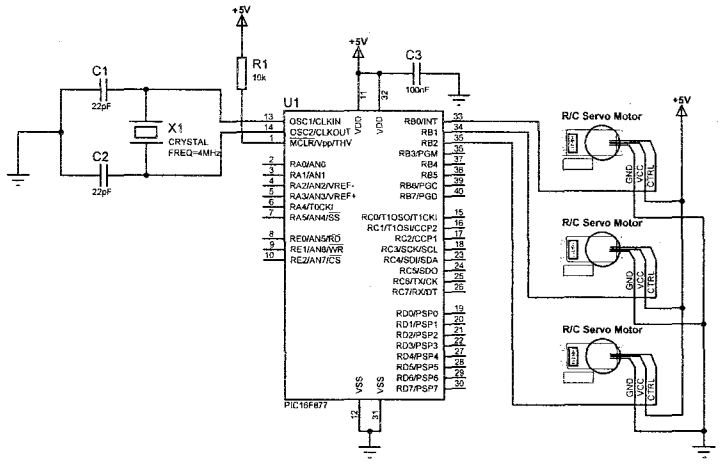
R/C servo motorlar genelde uzaktan kumandalı model uçak ve model gemi maketlerinde kullanılmaktadır. Bu motorlar genelde merkezde, tam solda ve tam sağda olacak şekilde yönlendirilirler. Ara açı değerlerinin bu motorlarda çok hassas olarak elde edilmesi zordur. R/C servoların dijital olanları da vardır. **Dijital R/C servo motorlar,** normal servo motorlara göre daha küçük açı değerlerinde de çalıştırılabilir. Bazı güç kaynaklarına fazla sayıda R/C servo motor bağlandığında bu motorların çalışması esnasında güç kaynağının voltajı dalgalanmaktadır. Bu dalgalanma PIC denetleyiciyi etkileyebilir. Bu nedenle uygulamalarınızda R/C servo motorları ayrı bir güç kaynağı ile beslemek daha iyi olmaktadır.

20.3.1. R/C Servo Motor Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada 3 adet R/C servo motorun aynı anda ayrı ayrı derecelerde dönmeleri sağlanmıştır. R/C servo motorların kontrolü için gerekli PWM sinyali PIC denetleyici içinde bulunan CCP modülünde PWM birimi ile sağlanabilir. Ama çoğu PIC'de 1 veya 2 adet CCP modülü bulunmakta veya hiç bulunmamaktadır. Bu da fazla sayıda R/C servo motorun kontrolü için yeterli değildir. Bu nedenle oluşturacağınız bir algoritma fazla sayıda servo motoru kontrol edebilmeniz gerekmektedir. Bu uygulamada oluşturulan bir algoritma ile 3 adet R/C servo motor aynı anda farklı derecelerde kontrol edilebilmektedir. İstenirse daha fazla R/C servo motor da kullanılabilir.

Devrenizin çalışmasını PIC PROG/DEKA kartında görmek için B Port'u çıkış terminalinden zil teli ile ilgili çıkışları R/C servo motorunuzun kontrol ucuna bağlamanız ve motora besleme gerilimlerini vermeniz yeterlidir. R/C servo konnektöründe 3 uç bulunur, + besleme, şase ve PWM kontrol (CTRL) sinyal ucu.



Şekil-20.14. *R/C Servo motor kontrol uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      PIC16F877 İle R/C Servo Motor Uygulaması
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları

#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.

// R/C Servo motorları dönme açı değerleri
const int8 servo_derece_1[]={8,9,10,11,12,13,14,15,16,17,18};
const int8 servo_derece_2[]={18,17,16,15,14,13,12,11,10,9,8};
const int8 servo_derece_3[]={8,13,18,8,13,18,8,13,18,8,13};

int i=0,pwm=0,duty_0=0,duty_1=0,duty_2=0;
int16 zaman=0; // 16 bit'lik değişken tanımlanıyor

#int_timer0  // Timer0 taşma kesmesi
void kesme ()
{
   set_timer0(217);  // TMR0 kaydedicisine 217 değeri yükleniyor
   if (pwm==0)       // Eğer PWM değişkeni 0 ise, 20msn dolmuş ise
   {
      output_high(pin_b0); // RB0 çıkışı lojik-1
      output_high(pin_b1); // RB1 çıkışı lojik-1
      output_high(pin_b2); // RB2 çıkışı lojik-1
   }

   if (pwm>=duty_0) output_low(pin_b0);
```

```
   if (pwm>=duty_1) output_low(pin_b1);
   if (pwm>=duty_2) output_low(pin_b2);

   zaman++; // zaman değişkenini 1 arttır

   // Servo motor dönüş adımları arası bekleme süresi için
   if (zaman>17350) // 115us*17350 = 1,995,250us yaklaşık 2msn
   {
      zaman=0;  // zaman değişkenini sıfırla
      i++;      // i değişkeni değerini 1 arttır
      if(i==11) // Eğer i değeri 11 ise-Tüm adımlar bitti ise
         i=0;   // i değişkenini sıfırla
   }

   pwm++;       // pwm değişkenini 1 arttır
   if (pwm>173) // pwm değeri 173'den büyük ise
      pwm=0;    // pwm değişkenini sıfırla
}

/********* ANA PROGRAM FONKSİYONU********/
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   // Timer0 ayarları belirtiliyor
   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_2);
   set_timer0(217);  // TMR0 kaydedicisine 217 değeri yükleniyor

   enable_interrupts(int_timer0); // Timer0 taşma kesmesi aktif
   enable_interrupts(global); // Aktif edilen tüm kesmelere izin ver

   output_b(0x00);  // İlk anda B port'u çıkışı sıfırlanıyor

   while(1)  // Sonsuz döngü
   {
      duty_0=servo_derece_1[i]; // 1. R/C servo PWM görev çevrimi
      duty_1=servo_derece_2[i]; // 2. R/C servo PWM görev çevrimi
      duty_2=servo_derece_3[i]; // 3. R/C servo PWM görev çevrimi
   }
}
//*****************************************************************
```

Programda R/C servo motorlara gönderilecek PWM kontrol sinyalleri için Timer0 zamanlayıcısı biriminden faydalanılmıştır. Timer0 zamanlayıcısı `setup_timer_0(RTCC_INTERNAL | RTCC_DIV_2);` komutu ile Timer0

zamanlayıcısının, denetleyicinin osilatör frekansı ile besleneceği ve bölme oranı değeri 2 olarak ayarlanmıştır. `set_timer0(217);` komutu ile de TMR0 kaydedicisine 217 değeri yüklenmiştir. Böylece aşağıda hesaplanan şekilde her 78µs'de bir Timer0 taşma kesmesi meydana gelir.

Kesme Süresi $=T_{osc}$x(Bölme Oranı)x(256-TMR0'a yazılan değer)

$$= \frac{1}{f_{osc}} \text{x2x(256-217)} = \frac{1}{1\text{Mhz}} \text{x2x39} = 1\mu\text{sx78} = 78\mu\text{s}$$

($f_{osc} = \frac{\text{Denetleyici Osilatör Frekansı}}{4} = \frac{4\text{ Mhz}}{4} = 1\text{Mhz}$ olduğunu hatırlayınız.)

Program kesme fonksiyonuna gittiğinde kesme fonksiyonundaki komutlardan dolayı kesme fonksiyonu içinde de belli bir süre gecikme yaşanmaktadır. CCS C programında Compile menüsünde "C/ASM List" komutu seçildiğinde C ile yazdığınız programın Assembly komut karşılığı önünüze gelecektir. Bu dosyada her C komutunun assembly karşılığı görülmektedir. Buradan kesme fonksiyonunda belirtilen komutların kaç adet assembly komuta karşılık geldiği sayılır. Denetleyicide kullanılan kristal frekansı 4Mhz'dir. Denetleyici bu frekansı 4'e böler ve 1Mhz elde edilir. Sonuçta denetleyici osilatör sinyal periyodu 1/1Mhz=1µsn elde edilir. Bu programda her assembly komutunun 1µsn süre de icra edileceği anlamına gelir. Böylece kesme fonksiyonunda saydığınız assembly komutu sayısı kadar µsn'lik bir süre kesme fonksiyonunda geçer. Kesme fonksiyonun komutlarının assembly komut karşılıklarını sayarken dikkat edilirse tüm komutlar her zaman işleme girmemektedir. Örneğin kesme fonksiyonundaki `if (pwm==0)` komutunda, eğer pwm değişkeni 0 ise döngü komutları işleyecektir. Onun haricin de döngü komutları işlemeyecektir. Dolayısıyla kesme fonksiyonundaki komutları sayarken bu husus da göz önüne alınmalıdır. Programda kesme fonksiyonunda yaklaşık 37µsn süre geçmektedir. Bu süreyi Timer0 taşma kesmesi süresine eklersek 78µsn+37µsn=115µsn eder. Böylece her kesme meydana geldiğinde 115µsn'lik bir süre geçer. Kesme fonksiyonunda `pwm++;` komutu ile her kesmede pwm değişkeni bir arttırılmaktadır. `if (pwm>173)` ve `pwm=0;` komutları ile pwm değişkeni 173'den büyük olunca sıfırlanmaktadır. Böylece 115µsnx173=19895µsn≈20msn eder. Böylece 20msn olunca pwm değişkeni sıfırlanmaktadır.

Timer0 kesme fonksiyonunda ilk anda pwm değişkeni 0 olduğundan `if (pwm==0)` döngüsü işleme girmektedir. Bu döngüde `output_high(pin_b0);`, `output_high(pin_b1);`, `output_high(pin_b2);` komutları ile motorlara bağlı B port'u pin'leri çıkışları lojik-1 yapılmaktadır. `if (pwm>=duty_0) output_low(pin_b0);`, `if (pwm>=duty_1) output_low(pin_b1);`, `if (pwm>=duty_2) output_low(pin_b2);` komutlarıyla istenen PWM sinyali görev çevrimi süresi sonunda motor girişlerine giden pin'lerin (RB0, RB1 ve RB2) çıkışları lojik-0 yapılmaktadır. Böylece örneğin RB0 çıkışı lojik-1 iken 1msn sonra lojik-0 yapılmıştır. Böylece RB0 çıkışındaki PWM sinyalinin frekansı 50Hz, görev çevrimi ise

1 msn olur. Yani RB0 çıkışındaki motor tam sola döner. Bu şekilde istenen görev çevrimi süreli PWM sinyali elde edilerek R/C servo motor kontrol edilir.

Programın başında dikkat edilirse `const int8 servo_derece_1[] = {8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18};` komutu gibi 3 adet komut kullanılmıştır. Bu komutlarla, R/C servo motorların dönmesi istenen derece değerleri tam sayı tipinde sabit olarak dizilere yüklenmiştir. Bu değerler sonsuz döngüdeki `duty_0=servo_derece_1[i];` komutlarıyla sırayla duty_0 ve diğer duty_1, duty_2 değişkenlerine yüklenmektedir. Peki, bu değerler nasıl belirlenmektedir? R/C servo motorlar 1msn ile 2msn'lik arası görev çevrimli 50Hz'lik sinyallerle çalışmaktaydı. Her kesme 115µsn sürmekte olduğundan, 1msn görev çevrimi için 1msn=1000µsn/115µsn≈8 değeri elde edilir. 2msn görev çevrimi için 2ms=2000µsn/115µsn ≈18 değeri elde edilir. Böylece -90^0 ile $+90^0$ arası hareket eden R/C servo motor için 8 ile 18 değerleri arası değerler servo_derece_1[], servo_derece_2[], servo_derece_3[] dizilerine yüklenerek istenen dönüşler elde edilir. Bu değerler ile R/C servo motor yaklaşık 20^0'lik açı değerleri ile döndürülebilir. Her dönüş değeri arası belli bir süre beklenmelidir. Bu süre kullanılan R/C servo motorun hızına bağlıdır. Programda her dönüş açısı için belli bir süre beklenmektedir. `zaman++;` komutu ile her kesmede zaman değişkeni bir arttırılmaktadır. `if (zaman>17350)` komutu ile zaman değişkeni 17350'den büyük olunca if döngüsünün işlem görmesi sağlanmıştır. Böylece 17350x115µsn=1.995.250µsn≈2sn'lik bir süre sonunda `if (zaman>17350)` komutu ile belirtilen döngü işleme girecek ve döngüde ilk başta `zaman=0;` komutu ile zaman değişkeni sıfırlanacaktır. Daha sonra ise `i++;` komutu işe i değişkeni değeri 1 arttırılmaktadır. Böylece yaklaşık her 2sn'de i değişkeni artmakta ve `servo_derece_1[i]` komutuyla motorlara gönderilen dönme açı değeri dizinin diğer elemanına geçmektedir. Bu döngüde aynı zamanda `if(i==11)` komutu ile i değerinin 11'e ulaşıp ulaşmadığı kontrol edilmiştir. 11 sayısı ilk başta tanımlanan R/C servo motorların dönme açı değerleri dizilerindeki toplam adım sayısıdır. Bu komut ile tüm adımların bitip bitmediği kontrol edilmektedir. Eğer tüm adımlar gerçekleştirilmişse `i=0;` komutu ile i değeri sıfırlanmakta ve motorlar tekrar dizinin başındaki dönüş açı değerlerinden dönmeye başlamaktadır. Dizideki adım sayısı uygulamanın özelliğine göre arttırılabilir.

Programın anlaşılması zor gibi görünse de, anlatılan bilgiler dikkatlice incelendiğinde az bir gayret ile programın mantığı rahatça anlaşılabilir.

21. Bölüm

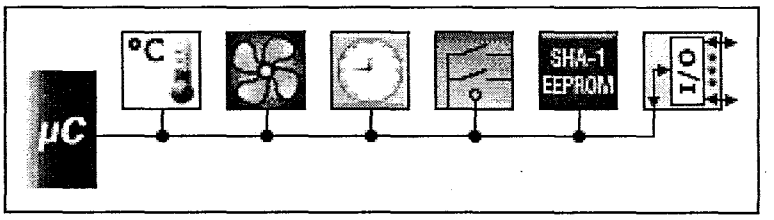
1-WIRE (TEK HAT) İLETİŞİM

- 1-Wire (Tek Hat) İletişim
- 1-Wire İletişim İçin Kendi Fonksiyonlarımızı Oluşturma
- DS1990A i-Button
- 1-Wire İletişim Uygulaması (DS1990A i-Button İle)
- CCS C 1-Wire İletişim Kütüphane Dosyası

Bu bölümde 1 Wire iletişimi protokolü ve işlemleri anlatılmıştır. Ardından 1 Wire iletişimi için CCS C'de fonksiyonların oluşturulması anlatılmıştır. 1 Wire iletişim uygulaması için Maxim-Dallas firmasının DS1990A i-button elemanı anlatılmış ve bir uygulama gerçekleştirilmiştir. En son olarak da CCS C derleyicisinde 1-Wire iletişim için bulunan kütüphane dosyası tanıtılmıştır.

21.1. 1 WIRE (TEK HAT) İLETİŞİMİ

1 Wire (Tek Hat) iletişimi Maxim-Dallas firmasının bazı ürünleri için geliştirdiği tek hat üzerinden hem çift yönlü bilgi alış-verişi hem de güç iletiminin sağlanabildiği bir iletişim metodudur. Master olan denetleyici birden fazla slave cihazı tek hat üzerinden kontrol edebilir.



Şekil-21.1. *1 Wire iletişim ile birden fazla cihaz kontrolü.*

1 Wire iletişime sahip tüm cihazların kendilerine özgü 64 bit'lik bir kayıt (registration) kodu bulunmaktadır. Bu kod ROM hafızada saklanmaktadır. Kodun düşük değerlikli ilk 7 bit'i cihazın **family (aile) kodunu** içerir. Aile kodu cihazın tanımlanması için kullanılır. Bu kod ile sistemde data hattına bağlı cihazların ne olduğu öğrenilebilir. Tablo-21.1'de 1-Wire iletişime sahip çeşitli entegreler ve aile

(family) kodları görülmektedir. Sonraki 48 bit ise cihazın seri numarasını (serial number) içerir. En sondaki 8 bit ise CRC (**C**yclic **R**edundancy **C**heck - Çevrimsel Hata Denetimi) kodunu içerir. Bu kod master cihazın, slave cihazdan yaptığı okumanın hatasız olup olmadığını denetlemesine izin verir.

MSB	64 Bit Kayıt (Registration) ROM Numarası	LSB
8 Bit CRC Kodu MSB LSB	48 Bit Seri Numara MSB LSB	8 Bit Aile Kodu (Family Code) MSB LSB

Şekil-21.2. *1 Wire cihazlarının kendilerine özgü 64 bit'lik ROM numarası bit'leri.*

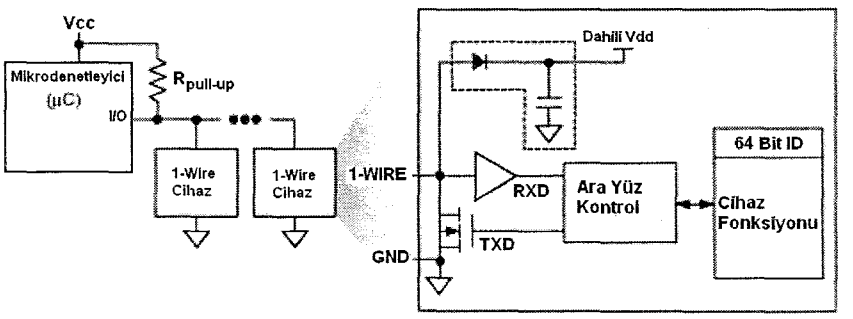
Cihaz Aile Kodu	Eleman İsmi	Açıklaması
01	(DS1990A), (DS1990R), DS2401, DS2411	i-Button
02	(DS1991), DS1425	Multikey i button, 1152-bit güvenli hafıza
04	(DS1994), DS2404	4kb NV RAM hafıza ve saat, zamanlayıcı, alarm
05	DS2405	Tekli adreslenebilir anahtar
06	(DS1993)	4kb NV RAM Hafıza
08	(DS1992)	1kb NV RAM Hafıza
09	(DS1982), DS2502	1kb EEPROM Hafıza
0A	(DS1995)	16kb NV RAM Hafıza
0B	(DS1985), DS2505	16kb EEPROM Hafıza
0C	(DS1996)	64kb'a kadar NV RAM Hafıza
0F	(DS1986), DS2506	64kb EEPROM Hafıza
10	(DS1920), DS1820, DS18S20	Alarmlı Sıcaklık Ölçme Sensörü
12	DS2406, DS2407	1kb EEPROM Hafıza, 2 kanal adreslenebilir anahtar
14	(DS1971), DS2430A	256 bit EEPROM Hafıza
18	(DS1963S)	4kb NV RAM Hafıza
1A	(DS1963L)	4kb NV RAM Hafıza
1C	DS28E04-100	4096 bit EEPROM Hafıza, 2 kanal adreslenebilir anahtar
1D	DS2423	Dış sayıcı ile 4kb NV RAM Hafıza
1F	DS2409	2 Kanal Adreslenebilir MikroLAN Bağlayıcı
20	DS2450	4 Kanal A/D Çevirici (ADC)
21	(DS1921), (DS1921H), (DS1921Z)	Thermochron i-button (Termometre, RTC)
22	DS1822	Dijital Sıcaklık Sensörü
23	(DS1973), DS2433	4kb EEPROM Hafızası
24	(DS1904), DS2415	Real Time Clock (Gerçek Zamanlı Saat-Takvim)
26	DS2438	Akıllı Pil-Akü Monitörü (Smart Battery Monitor)
27	DS2417	Kesmeli RTC (Gerçek Zamanlı Saat-Takvim)

28	DS18B20	Dijital Sıcaklık Sensörü
29	DS2408	8 Kanal Adreslenebilir Anahtar
2C	DS2890	Tek Kanal Dijital Potansiyometre
2D	DS2431	1024 Bit EEPROM Hafıza
30	DS2760	Sıcaklık, Akım,ADC
33	(DS1961S), DS2432	1k EEPROM Hafıza
37	(DS1977)	Şifre Korumalı 32KByte EEPROM
3A	(DS2413)	İki Kanallı Adreslenebilir Anahtar
41	(DS1922L), (DS1922T), (DS1923), DS2422	Yüksek Kapasiteli Sıcaklık ve Nem Kaydedicisi

()- Parantez içine alınan cihazlar i-Button yapısındadır.

Tablo-21.1. *1-Wire iletişime sahip cihazlar ve aile (family) kodları.*

Mikrodenetleyici ile kontrol edilen 1 Wire cihazları için data hattı mutlaka pull-up direnci ile + beslemeye (Vdd) bağlanmalıdır. Bu data hattı hem çift yönlü data iletişimi için kullanıldığı gibi aynı zamanda bazı 1 Wire cihazları için besleme gerilimi temin hattıdır. Data hattından beslenen 1 Wire cihazları içinde bir kondansatör vardır. Data hattı lojik-1 (5 V) seviyesinde iken bu kondansatör şarj olur. Kondansatör üzerindeki şarj gerilimini kullanarak cihaz kendi enerjisini sağlar. 1 Wire iletişimin standart hızı 15kbps (kilo bit per second)'dir. 1 Wire iletişiminin yüksek hızı ise yaklaşık 115kbps'dir.

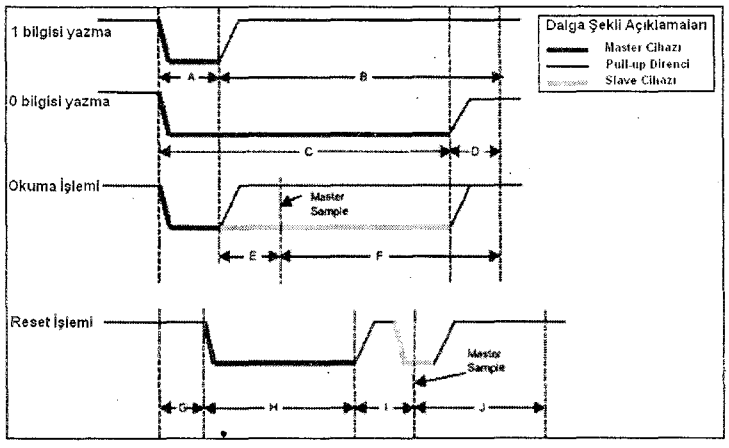


Şekil-21.3. *Mikrodenetleyicinin 1 Wire cihazlara bağlantısı ve 1 Wire cihazı iç yapısı.*

21.1.1. 1-Wire İletişim İşlemleri

1 Wire iletişimde 1 Wire cihazlarına reset sinyali gönderme, bilgi okuma ve bilgi yazma işlemleri belli bir protokole göre yapılır. Bu protokolde 4 temel işlem vardır. Bu işlemler **reset, Bir bit "1" bilgisi yazma, Bir bit "0" bilgisi yazma, 1 bit veri okuma** işlemleridir. Bu işlemleri gerçekleştirmek için data hattını belli bir süre lojik-1(High), belli bir süre lojik-0 (Low) seviyesinde tutmak gerekir. Bu süreler µsn düzeyindedir. Şekil-21.4'de 1-Wire iletişimde gerçekleştirilecek işlemler için gerekli sinyallerin dalga şekilleri verilmiştir. Bu dalga şekillerinde istenen işlem için gerekli süreler A, B, C,...,J harfleri ile belirtilmiştir. Sürelere denk gelen bu harflerin süre

karşılıkları Tablo-21.2de verilmiştir. Tablo-21.2'de verilen süre değerleri için minimum ve maksimum değerleri belirtilmiştir. İstenen işlem için bu sürelerin dışına çıkılmaması gerekir. Tablo-21.3'de de Şekil-21.4'deki sinyallerin açıklaması olarak temel 1-Wire İletişim işlemlerinin nasıl yapılması gerektiği verilmiştir.



Şekil-21.4. *1-Wire İletişim işlemleri sinyal dalga şekilleri.*

Parametre	İletişim Hızı Türü	Gerekli Süre		
		Minimum	Tavsiye Edilen	Maksimum
A	Standart Hız	5µsn	6µsn	15µsn
	Yüksek Hız	1µsn	1.5µsn	1.85µsn
B	Standart Hız	59µsn	64µsn	-
	Yüksek Hız	7.5µsn	7.5µsn	-
C	Standart Hız	60µsn	60µsn	120µsn
	Yüksek Hız	7µsn	7.5µsn	14µsn
D	Standart Hız	8µsn	10µsn	-
	Yüksek Hız	2.5µsn	2.5µsn	-
E	Standart Hız	5µsn	9µsn	12µsn
	Yüksek Hız	0.5µsn	0.75µsn	0.85µsn
F	Standart Hız	50µsn	55µsn	-
	Yüksek Hız	6.75µsn	7µsn	-
G	Standart Hız	0µsn	0µsn	0µsn
	Yüksek Hız	2.5µsn	2.5µsn	-
H	Standart Hız	480µsn	480µsn	640µsn
	Yüksek Hız	68µsn	70µsn	80µsn
I	Standart Hız	63µsn	70µsn	78µsn
	Yüksek Hız	7.2µsn	8.5µsn	8.8µsn
J	Standart Hız	410µsn	410µsn	-
	Yüksek Hız	39.5µsn	40µsn	-

Tablo-21.2. *1-Wire İletişim işlemleri dalga sinyalleri parametre süreleri.*

Tablo-21.2'de verilen süreler 1-Wire iletişimde genel olarak kullanılan işlem sürelerini belirtir. Hassas işlemler için gerekli süreleri kullandığınız elemanın teknik bilgi sayfalarında (datasheet) bulabilirsiniz. Tablo-21.2'de hem standart hem de hızlı 1-Wire iletişim için gerekli süreler verilmiştir. Genelde standart ve tavsiye edilen süre değerleri kullanılır.

İşlem	Açıklama	Yapılması Gereken İşlemler
1 Bit "1" Bilgisi Yazma	Slave cihaza 1 bit "1" bilgisi gönderilir.	Data hattı lojik-0 yapılır, A süresi kadar gecikme verilir, Data hattı boşta veya lojik-1 yapılır, B süresi kadar gecikme verilir.
1 Bit "0" Bilgisi Yazma	Slave cihaza1 bit "0" bilgisi gönderilir.	Data hattı lojik-0 yapılır, C süresi kadar gecikme verilir, Data hattı boşta veya lojik-1 yapılır, D süresi kadar gecikme verilir.
1 Bit Okuma İşlemi	Slave cihazdan 1 bit bilgi okunur.	Data hattı lojik-0 yapılır, A süresi kadar gecikme verilir, Data hattı boşta veya lojik-1 yapılır, E süresi kadar gecikme verilir, data hattı okunur, F süresi kadar gecikme verilir.
Reset Sinyali	1-Wire cihazlarını resetler ve onları komut alımına hazır hale getirir.	G süresi kadar gecikme verilir, Data hattı lojik-0 yapılır, H süresi gecikme verilir, Data hattı boşta veya lojik-1 yapılır, I süresi kadar gecikme verilir, data hattı okunur, J süresi kadar gecikme verilir.

Tablo-21.3. *1-Wire iletişim işlemleri ve gerçekleştirilme aşamaları.*

21.1.1.1. Reset Sinyali Gönderme

1-Wire iletişimde ilk başta mutlaka reset sinyali master cihaz tarafından gönderilmelidir. Bu işlem için standart hızda data hattı G süresi kadar (0 µsn) (Yüksek hızda 2.5 µsn) lojik-1 iken, data hattı çıkışı denetleyici tarafından 480 µsn kadar (H süresi kadar) lojik-0 yapılır. Daha sonra 70 µsn kadar (I süresi kadar) data hattı boşta bırakılır veya lojik-1 yapılır. En son olarak da data hattı denetleyici tarafından okunur. Okunan bit "0" ise data hattının öbür ucunda reset sinyaline cevap veren bir cihaz olduğu anlamına gelir. Eğer okunan bit "1" ise bu değerin zaten pull-up direnci sayesinde Vdd seviyesinde olan data hattının seviyesi olduğu anlaşılır. Bu da bize karşı tarafta cihaz olmadığını veya cihaz olsa dahi yanıt vermediğini ya da herhangi bir nedenle veremediği bilgisini verir.

21.1.1.2. Bir Bit 1 Bilgisi Yazma

Bu işlem için ilk başta data hattı 6 µsn kadar (A süresi kadar) lojik-0 yapılır. Ardından data hattı 64 µsn kadar (B süresi kadar) boşta bırakılır veya lojik-1 yapılır. Böylece bir bit 1 bilgisi karşı tarafa gönderilmiş (yazılmış) olunur.

21.1.1.3. Bir Bit 0 Bilgisi Yazma

Bu işlem için ilk başta data hattı 60 µsn kadar (C süresi kadar) lojik-0 yapılır. Ardından data hattı 10 µsn kadar (D süresi kadar) boşta bırakılır veya lojik-1 yapılır. Böylece bir bit 0 bilgisi karşı tarafa gönderilmiş (yazılmış) olunur.

21.1.1.4. Bir Bit Okuma İşlemi

Bu işlem için ilk başta data hattı 6 µsn kadar (A süresi kadar) lojik-0 yapılır. Daha sonra data hattı 9 µsn kadar (E süresi kadar) boşta bırakılır veya lojik-1 yapılır. Ardından data hattı okunur. Data hattı okunduktan sonra 55 µsn kadar (F süresi kadar) beklenir ve işlem sonlandırılır.

21.2. 1-WIRE İLETİŞİM İÇİN KENDİ FONKSİYONLARIMIZI OLUŞTURMA

1-Wire iletişim için CCS C'de hazır fonksiyonlar olmasına rağmen konuyu daha iyi anlamak ve hâkim olmak amacıyla ilk başta 1-Wire işlemlerinin CCS C komutlarını kullanarak kendimiz tarafından oluşturulması anlatılmıştır. Her programda 1-Wire iletişim için aynı komutları tekrar tekrar yazmak gereksizdir. 1-Wire işlemlerini yerine getiren fonksiyonlar yazmak bize tüm programlarımızda istediğimiz zaman bu fonksiyonları çağırarak kullanmamızı sağlar. Aşağıda bu işlemler için ayrı ayrı fonksiyonlar yazılmıştır. **Bu fonksiyonları aynı dosya içine kaydederek (örneğin 1_wire.c adında) istediğimiz programda bu dosyayı çağırarak tüm 1-Wire iletişimi fonksiyonlarını kullanabiliriz. Bu dosyayı #include <1_wire.c> komutu ile programa eklemeniz gerekmektedir. Bu dosya içinde gecikme komutları kullanıldığından bu komutu #use delay (clock=4000000) komutundan sonra kullanmanız gerekmektedir. Bu dosyayı programınızda kullanmak için C:\Program Files\PICC\Drivers\ dizinine kopyalamayı unutmayınız.**

Bu dosyanın en üstüne 1-Wire iletişim için kullanılacak data pin'i ismi belirtilmelidir. Ayrıca oluşturulacak bu dosyada kullanılacak bazı genel değişkenlerin tanımlanması gereklidir. Bu dosyanın başına aşağıda komutlar mutlaka yazılmalıdır.

```
#define ow_data_pin pin_c0 // 1-Wire iletişimindeki data pin'i
int16 A,B,C,D,E,F,G,H,I,J; // Kullanılan genel değişkenler
```

21.2.1. 1-Wire İletişim Hızını Seçme Fonksiyonu

Bu fonksiyonda 1-Wire iletişim hızını seçme fonksiyonu oluşturulmuştur. Fonksiyon "1" bilgisi ile çağrıldığında iletişim için standart hız, fonksiyon "0" bilgisi ile çağrıldığında iletişim için yüksek hız seçilmiş demektir. Yüksek hızda Tablo-21.2'de

verilen yüksek hız değerlerinin bazıları ondalıklıdır. Ondalıklı değerler ile hesaplama denetleyici de çok fazla süre kaybedeceğinden bu değerler yuvarlatılmıştır.

```
//***** İletişim Hızı Seçme Fonksiyonu *****
void ow_hiz(int hiz)
{
   if (hiz==1) // hiz değeri "1" ise standart hız geçerli
   {
      A=6; B=64; C=60; D=10; E=9; F=55; G=0; H=480; I=70; J=410;
   }

   if (hiz==0) // hiz değeri "0" ise Yüksek hız geçerli
         // Yüksek hızdaki ondalıklı süre değerleri yuvarlatılmıştır
   {
      A=1; B=7; C=7; D=2; E=1; F=7; G=2; H=70; I=8; J=40;
   }
}
```

21.2.2. 1-Wire İletişim Reset Sinyali Fonksiyonu

Bu fonksiyonda 1-Wire iletişimin reset sinyalini oluşturan fonksiyon CCS C derleyicisi komutları ile yazılmıştır. Fonksiyon "1" veya "0" değeri ile geri döner. Geri dönüş değeri "1" ise data hattında bir 1-Wire cihazının olmadığı veya cevap vermediği anlamına gelir. Fonksiyon geri dönüş değeri "0" ise data hattında 1-Wire cihazının olduğu ve sinyale yanıt verdiği anlamına gelir.

```
//***** 1-Wire İletişim Reset Sinyali Gönderme Fonksiyonu *****
int1 ow_reset(void)
{
   int1 sonuc; // sonuc adında 1 bit'lik bir değişken tanımlanıyor

   delay_us(G); // G süresi kadar gecikme veriliyor
   output_low(ow_data_pin); // 1-Wire data pin'i çıkışı lojik-0
   delay_us(H); // H süresi kadar gecikme veriliyor
   output_float(ow_data_pin); // 1-Wire data pin'i boşta (giriş
                              // olarak yönlendirildi)
   delay_us(I); // I süresi kadar gecikme veriliyor
   sonuc=input(ow_data_pin); // 1-Wire data pin'i okunuyor
   delay_us(J); // J süresi kadar gecikme veriliyor

   return sonuc; // Fonksiyon sonuc değişkeni değeri ile geri döner
}
```

21.2.3. 1-Wire İletişimde 1 Bit Bilgi Yazma Fonksiyonu

Bu fonksiyonda, 1-Wire iletişim de data hattına 1 bit'lik "1" veya "0" bilgisi yazma işlemleri gerçekleştirilmiştir.

```
//***** 1-Wire İletişimde Bir Bit Bilgi Yazma Fonksiyonu *****
void ow_yaz_bit(int bit)
{
   if (bit==1) // "1" Bilgisi Yazılır
   {
      output_low(ow_data_pin); // 1-Wire data pin'i çıkışı lojik-0
      delay_us(A); // A süresi kadar gecikme veriliyor
      output_float(ow_data_pin); // 1-Wire data pin'i boşta (giriş
                                 // olarak yönlendirildi)
      delay_us(B); // B süresi kadar gecikme veriliyor
   }
   else  // "0" Bilgisi Yazılır
   {
      output_low(ow_data_pin); // 1-Wire data pin'i çıkışı lojik-0
      delay_us€; // C süresi kadar gecikme veriliyor
      output_float(ow_data_pin); // 1-Wire data pin'i boşta (giriş
                                 // olarak yönlendirildi)
      delay_us(D); // D süresi kadar gecikme veriliyor
   }
}
```

21.2.4. 1-Wire İletişimde 1 Bit Bilgi Okuma Fonksiyonu

Bu fonksiyonda, 1-Wire iletişim de data hattından 1 bit'lik bilgi okumak için gerekli işlemler gerçekleştirilmiştir.

```
//***** 1-Wire İletişimde Bir Bit Bilgi Okuma *****
int1 ow_oku_bit(void)
{
   int1 bit; // Bir bit'lik değişken tanımlanıyor

   output_low(ow_data_pin); // 1-Wire data pin'i çıkışı lojik-0
   delay_us(A); // A süresi kadar gecikme veriliyor
   output_float(ow_data_pin); // 1-Wire data pin'i boşta (giriş
                              // olarak yönlendirildi)
   delay_us€; // E süresi kadar gecikme veriliyor
   bit=input(ow_data_pin);
   delay_us(F); // F süresi kadar gecikme veriliyor

   return bit; // Fonksiyon bit değişkeni değeri ile geri döner
}
```

21.2.5. 1-Wire İletişimde 1 Byte Bilgi Yazma Fonksiyonu

Bu fonksiyon ile önceden oluşturulmuş, 1-Wire iletişim de bir bit'lik bilgi yazma fonksiyonu **ow_yaz_bit()** kullanılarak 1-Wire cihazına bir byte'lık bir bilgi yazma işlemini gerçekleştiren fonksiyon yazılmıştır. Fonksiyonda gönderilecek 1 byte (8 bit) bilgi değeri bit bit gönderilmektedir. Fonksiyondaki `bit_test(data,i)` komutu ile data değişkeni değerinin bit'leri sırayla (i değeri ile) okunmaktadır. Fonksiyonda gönderilecek bilginin LSB bit'i ilk önce gönderilmektedir.

```
//***** 1-Wire İletişimde Bir Byte Bilgi Yazma Fonksiyonu *****
void ow_yaz_byte(int data)
{
   int i; // Okunacak bilginin bit sayısını tutan değişken

   for (i=0;i<8;i++) // Byte bilgisi bit bit yazdırılıyor. LSB bit'i
                     // ilk önce gönderiliyor.
   {
      ow_yaz_bit(bit_test(data,i)); // data değişkeni değerinin
                                    // bit'leri tek tek okunup gönderiliyor
   }
}
```

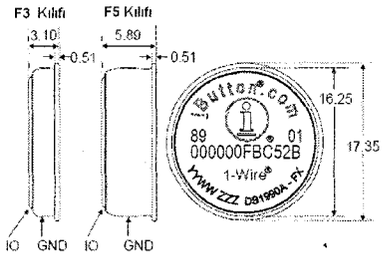
21.2.6. 1-Wire İletişimde 1 Byte Bilgi Okuma Fonksiyonu

Bu fonksiyon ile önceden oluşturulmuş, 1-Wire iletişim de bir bit'lik bilgi okuma fonksiyonu **ow_oku_bit()** kullanılarak 1-Wire cihazından bir byte'lık bir bilgi okuma işlemini gerçekleştiren fonksiyon yazılmıştır. Fonksiyonda 1-Wire cihazından bilgiler bit bit okunmaktadır. `if (ow_oku_bit())` komutu ile okunan bit değerinin "1" olup olmadığı kontrol edilmiştir. Eğer okunan bit değeri "1" ise `sonuc=(sonuc | 0x80);` komutu ile "sonuc" değişkeni değeri 0x80 değeri ile OR (VEYA) işlemine tabi tutulmaktadır. Böylece okunan bit değeri "1" olunca bu değer "sonuc" değişkeninin MSB (7.bit'i) bit'ine yazılmış olur. Fonksiyondaki `sonuc=sonuc>>1;` komutu ile de bu MSB değerleri 1 bit sağa kaydırılır. Bu işlemler `for (i=0;i<8;i++)` döngüsü ile 8 defa yapılır. Sağa kaydırma işleminde bilgiye 0 eklenerek kaydırma işlemi yapıldığını hatırlayınız. Böylece okunan bit değeri "0" ise fonksiyonda sadece `sonuc=sonuc>>1;` komutu aktif olur ve okunan "0" bilgisi "sonuc" değişkenine yazılmış olur.

```
//***** 1-Wire İletişimde Bir Byte Bilgi Okuma Fonksiyonu *****
int ow_oku_byte(void)
{
   int i,sonuc=0; // Tam sayı tipinde değişkenler tanımlanıyor

   for (i=0;i<8;i++)
   {
      sonuc=sonuc>>1;     // sonuc değişkenini 1 bit sağa kaydır

      if (ow_oku_bit())         // Eğer okunan bit "1" ise
         sonuc=(sonuc | 0x80); // Okunan bit'i sonuc değerinde
belirt
   }
   return sonuc;  // Okunan byte değeri ile geri dön
}
```

21.3. DS1990A i-BUTTON

DS1990A elemanı i-Button yapısında ve üretici firma tarafından sadece elemana özgü olarak elamanın ROM hafızasına kayıt numarası yüklenmiş bir data taşıyıcı olarak tanımlanabilir. DS1990A elemanı çeşitli uygulamalarda dijital anahtar olarak kullanılabilir. Bunun için bir denetleyici ile DS1990A elemanının içindeki kayıt numarası okunur ve sistem de kayıtlı numara ile karşılaştırılır. Eğer elemanın kodu ile sistem de bulunan kod aynı ise istenen işlemler gerçekleştirilir. Örneğin bir giriş kapısı kontrol işlemi bu şekilde yapılabilir. İstanbul'da şehir içi toplu taşıma araçlarında kullanılan **AKBİL** de bir i-button'dur. I-Button'ların hafızalı olanları da vardır. Böylece i-Button'a istenen bilgiler yüklenerek değiştirilebilir. AKBİL bu mantığa göre çalışmaktadır. Şekil-21.5'de DS1990A elemanının da sahip olduğu bir i-Button kılıf yapısı görülmektedir DS1990A iki farklı boyuta (F3 ve F5) sahip kılıflar da üretilmektedir. **DS1990A'da iki uç vardır. Birisi şase ucu GND, diğeri de 1-Wire iletişimde kullanılacak data ucu olan IO ucudur.**



Şekil-21.5. *DS1990A i-Button kılıfı ve boyutları (ölçü değerleri mm cinsindendir)*

DS1990A elemanına harici bir enerji vermek gerekmez. DS1990A enerjisini data hattından alır. DS1990A'yı devrede kullanırken data hattı yaklaşık 2.2KΩ pull-up direnci ile Vdd (pozitif besleme) hattına bağlanmalıdır.

DS1990A elemanında, diğer 1-Wire elemanlarında olduğu gibi ROM hafızasında 64 bit'lik kayıt numarası bulunur. Önceden bilindiği gibi bu 64 bit'in düşük değerlikli ilk **8 bit'i aile kodu**, diğer **48 bit seri numarası** ve en son **8 bit'i ise CRC** kodudur.

DS1990A elemanını kullanmak için ilk başta data hattına reset sinyali gönderilerek eleman kullanıma hazır hale getirilir. Ardından ROM fonksiyonu okuma komutları gönderilerek istenen işlemler yapılır. Örneğin 0x33 komutu DS1990A elemanının ROM hafızası bilgilerini okuma komutudur.

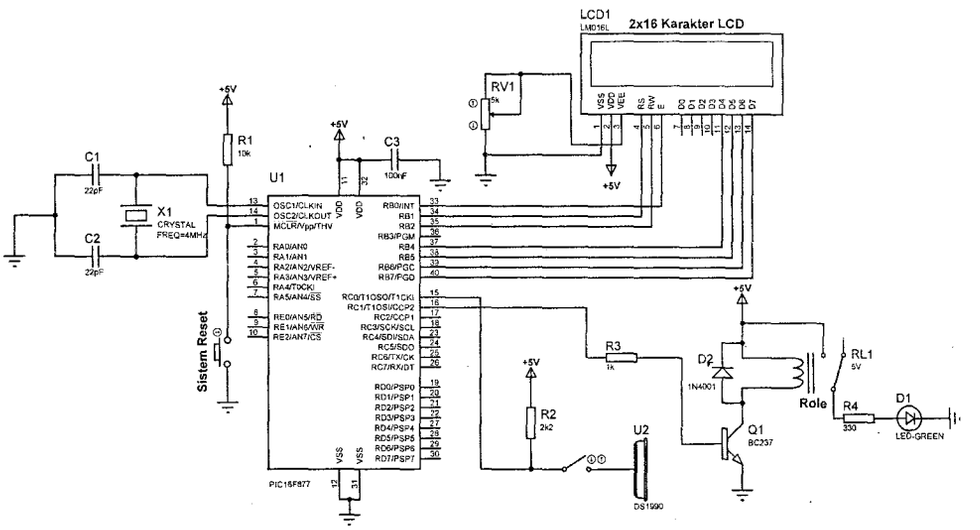
21.4. 1-WIRE İLETİŞİM UYGULAMASI (DS1990A İLE)

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada DS1990A ile hem uygulama da 1-Wire iletişim işlemleri incelenmiş hem de bir i-button'lu bir kapı kontrol uygulaması gerçekleştirilmiştir. Devrede DS1990A elemanı ile RC0 pin'inden iletişim kurulmuştur. Bu hat 2.2KΩ değerinde pull-up direnci ile + beslemeye bağlanmıştır. Data hattı bir anahtar ile DS1990A elemanına bağlanmıştır. Böylece DS1990A elemanının sisteme yerleştirilip yerleştirilmediği Proteus-ISIS programında belirtilebilir. Devre programında sisteme yerleştirilen DS1990A elemanının aile kodu, seri numarası ve CRC kodu okunarak LCD ekranda gösterilmektedir. Ayrıca RC1 pin'ine bir transistör ile ROLE bağlanmıştır. ROLE kontaklarına örnek olması için LED bağlanmıştır. Böylece sisteme yerleştirilen DS1990A elemanının ROM kayıt numarası eğer programda kayıtlı olan numara ile aynı ise ROLE enerjilendirilerek LED yanmaktadır. Siz uygulamanızda LED yerine başka bir eleman örneğin bir kapı kontak bobinini bağlayabilirsiniz.

Devre programında Bölüm-21.2'de tanımlanan fonksiyonlar "1_wire.c" adında bir dosyaya kaydedilmiş ve programa eklenmiştir. Bu dosyayı C:\Program Files\PICC\Drivers\ dizinine kopyalamayı unutmayınız.

Proteus programında benzetim yaparken DS1990A elemanının üzerine fare ile önce sağ sonra sol tuşa bastığınızda açılan ayar menüsünde "Rom Serial Number" kısmını istediğiniz gibi değiştirebilirsiniz. Böylece çalışmanızda LCD ekranda gözüken DS1990A seri numarasının değiştiğini görebilirsiniz.



Şekil-21.6. *1-Wire iletişim uygulaması (DS1990A ile) devre şeması.*

Devre programı;

```
/*****************************************************************
    PIC16F877 ile 1-Wire (Tek Hat) İletişim Uygulaması (DS1990A İle)
PIC PROG/DEKA      : Port B jumper'ı LCD konumunda olmalı
*****************************************************************/
#include <16f877.h>      // Kullanılacak denetleyicinin başlık
                         // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
#include <1_wire.c>   // 1-Wire iletişim fonksiyonlarının bulunduğu
                      // dosya programa ekleniyor

#define use_portb_lcd TRUE   // LCD B portuna bağlı
#include <lcd.c>             // lcd.c dosyası tanıtılıyor

int1 sonuc; // 1 bit'lik değişken tanımlanıyor
int  seri_no1=0,  seri_no2=0,  seri_no3=0,  seri_no4=0,  seri_no5=0,
seri_no6=0, seri_no7=0, seri_no8=0; //8 bit'lik değişkenler
int32 seri_no=0; // 32 bit'lik değişken tanımlanıyor

//********** ANA PROGRAM FONKSİYONU************
void main ( )
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   lcd_init();   // LCD hazırlanıyor

   ow_hiz(1);   // 1-Wire iletişim hızı standart seçildi

   while(1) // Sonsuz döngü
   {
      sonuc=ow_reset();  // 1-Wire cihazına reset sinyalini gönder

      if (sonuc==0)  //Data hattında 1-wire cihazı var ve sağlam ise
      {
         printf(lcd_putc,"\fCihaz calisiyor");
         delay_ms(1500); // 1.5 sn gecikme veriliyor

         ow_yaz_byte(0x33); // ROM numarası okuma komutu cihaza
                            // gönderiliyor
         seri_no1=ow_oku_byte(); //ROM numarasının ilk byte değeri
                                 // aile (family) kodu'dur.
```

```
        seri_no2=ow_oku_byte(); // ROM numarasının 2. byte ile
        seri_no3=ow_oku_byte(); // 7.byte değerleri cihazın 48
        seri_no4=ow_oku_byte(); // bit'lik  seri nosunu oluşturur
        seri_no5=ow_oku_byte();
        seri_no6=ow_oku_byte();
        seri_no7=ow_oku_byte();
        seri_no8=ow_oku_byte(); // ROM numarasının 8. byte değeri
                                // cihazın CRC değeridir

      // 8 bitlik değerleri birleştirerek 32 bitlik değer oluşturur
        seri_no=make32(seri_no2,seri_no3,seri_no4,seri_no5);
      //Aile kodu LCD'ye yazdırılıyor
        printf(lcd_putc,"\fCihaz Aile Kodu");
        printf(lcd_putc,"\n%X",seri_no1);
        delay_ms(2000); // 2 sn gecikme veriliyor

      // Cihaz seri no LCD'ye gönderiliyor
      printf(lcd_putc,"\fCihaz Seri No");
      printf(lcd_putc,"\n%X%X%X%X%X%X",seri_no2,seri_no3,seri_no4,
      seri_no5,seri_no6,seri_no7);
      delay_ms(2000); // 2 sn gecikme veriliyor

      // Cihazın CRC değeri LCD'ye gönderiliyor
        printf(lcd_putc,"\fCihaz CRC Degeri");
        printf(lcd_putc,"\n%X",seri_no8);
        delay_ms(2000); // 2 sn gecikme veriliyor

        if (seri_no==0x2CC5B800) // Eğer doğru cihaz sistemdeyse
           output_high(pin_c1);  // RC1 pin'ine bağlı role çeker
      }
      else // Data hattında 1-wire cihazı yok veya sağlam değilse
        printf(lcd_putc,"\fCihaz calismiyor");

      delay_ms(1500);  // 1.5 sn gecikme veriliyor
   }
}
//*****************************************************************
```

Programda `#include <1_wire.c>` komutu ile 1-Wire işlemleri için oluşturduğumuz fonksiyonların içinde bulunduğu dosya programa eklenmiştir. Ana fonksiyonda `ow_hiz(1);` komutu ile 1-Wire iletişim için standart hız değerlerinin kullanılacağı belirtilmiştir. Sonsuz döngüde `sonuc=ow_reset();` komutu ile reset sinyali data hattındaki DS1990A cihazına gönderilmiştir. Bu komut ile data hattında bir 1-Wire iletişime sahip cihazın olup olmadığı veya olsa bile çalışıp çalışmadığı kontrol edilerek, hattaki 1-Wire elemanı iletişime hazır hale getirilmektedir. Data hattında bir 1-Wire elemanı varsa fonksiyon "0" değeri ile geri döner. `if (sonuc==0)` komutu ile `ow_reset()` fonksiyonunun geri dönüş değeri kontrol edilmektedir. Fonksiyon geri dönüş değeri "0" ise döngü işlemektedir. Eğer fonksiyon geri dönüş değeri "1" ise else komutunun altındaki `printf(lcd_putc,"\fCihaz calismiyor");` komut işlenecek ve LCD'de "Cihaz

calismiyor" ifadesi yazacaktır.

`ow_yaz_byte(0x33);` komutu ile DS1990A elemanına komut gönderilmiştir. Bu gönderilen heksadesimal 0x33 komutu ile DS1990A elemanına ROM okuma komutu verilmiş olunur. Ardından `seri_no1=ow_oku_byte();` komutu ile 64 bit'lik ROM kodunun düşük değerlikli ilk 8 bit'i yani ilk byte değeri okunur. Bu değer elemanın aile (family) kodudur. `seri_no2=ow_oku_byte();`, `seri_no3=ow_oku_byte();`,`seri_no4=ow_oku_byte();`,`seri_no5=ow_oku_byte();`,`seri_no6=ow_oku_byte();`,`seri_no7=ow_oku_byte();` komutlarıyla elemanın ROM kodundaki 48 bit'lik seri numarası byte byte okunmaktadır. En son olarak da ROM kodunun son byte değeri (8 bit'i) `seri_no8=ow_oku_byte();` komutu ile okunmaktadır. Bu değer de elemanın CRC kodunu oluşturmaktadır. `seri_no=make32(seri_no2,seri_no3,seri_no4,seri_no5);` komutu ile DS1990A elemanının seri numarasını oluşturan 8 bit'lik değişkenler birleştirilerek 32 bit'lik bir değer elde edilmektedir. Normalde cihaz seri numarası 48 bit'tir. Biz bu örnekte cihaz seri numarasının 32 bit'ini kullandık.

Programda `printf(lcd_putc,"\n%X",seri_no1);` komutu ile DS1990A'nın aile kodu LCD'ye heksadesimal olarak (komutta ki %X kısmı ile) yazdırılmaktadır. `printf(lcd_putc,"\n%X%X%X%X%X%X",seri_no2,seri_no3,seri_no4,seri_no5,seri_no6,seri_no7);` komutu ile de LCD'ye DS1990A elemanının seri numarası heksadesimal olarak yazılmıştır. `printf(lcd_putc,"\n%X",seri_no8);` komutuyla da LCD'ye DS1990A elemanının CRC kodu heksadesimal olarak (komutta ki %X kısmı ile) yazılmıştır.

`if (seri_no==0x2CC5B800)` komutu ile data hattındaki DS1990A elemanının seri numarasının 0x2CC5B800 sayısı ile aynı olup olmadığı karşılaştırılmaktadır. Eğer data hattındaki elemanın seri numarası ile bu numara aynı ise `output_high(pin_c1);` komutu ile RC1 pin'i lojik-1 yapılmaktadır. Bu sayede RC1 pin'ine bağlı transistör iletime geçmekte, ROLE enerjilenmekte ve ROLE kontağı çekmektedir. ROLE kontağı çekince LED yanmaktadır. Siz LED yerine başka bir eleman bağlayarak istediğiniz elemanın kontrolünü sağlayabilirsiniz.

21.5. CCS C 1-WIRE İLETİŞİM KÜTÜPHANE DOSYASI

CCS C derleyicisi Dallas firmasının 1-Wire iletişime sahip cihazları için C:\Program Files\PICC\Drivers dizini içinde **"TOUCH.c"** adında bir dosya içerir. Bu dosya içinde standart hızda 1-Wire iletişim için gerekli reset, 1 byte yazma ve 1 byte okuma fonksiyonları tanımlanmıştır. Aşağıda bu dosya içinde bulunan fonksiyonlar ve açıklamaları verilmiştir.

ⓘ **Neden bu bölümün ilk kısımlarında bu hazır fonksiyonların tanıtılmadığı sorusu aklınıza gelebilir.** Bu kısma kadar olan konu başlıklarını

sırayla irdeleyerek incelediğinizde 1-Wire iletişim için gerekli bilgilere ve istediğiniz şekilde iletişimi yönlendirme becerisine hâkim olmuş olacaksınız. Ayrıca program algoritmalarınızda hazır fonksiyonları kullanmanın yanında kendi özel amaçlarınız ve elektronik elemanlarınız için sistem tasarımınıza özgü fonksiyonlarınızı da oluşturma becerisini geliştirmiş olacaksınız. Bu öğrendiğiniz bilgiler ve tecrübe ışığında ister hazır fonksiyonları isterseniz de kendi özel fonksiyonları kullanarak konuya tam olarak hâkim olmuş olursunuz.

değişken ismi = touch_present(); Fonksiyonu Bu fonksiyon 1-Wire iletişimde reset sinyali oluşturmaya yarar. Bilindiği gibi bu sinyal ile 1-Wire cihazının hatta olup olmadığı kontrol edilir ve iletişime hazırlanır. Bu fonksiyon eğer cihaz hatta ise ve cevap vermişse "1 (TRUE)" bilgisi ile aksi durumda "0" bilgisi ile geri döner.

```
sonuc=touch_present(); // sonuc değişkeni "1" ise 1-Wire cihazı var ve
                       // çalışıyor anlamındadır
```

değişken ismi = touch_read_byte(); Fonksiyonu Bu fonksiyon 1-Wire iletişimde slave olan elemandan 1 Byte bilgi okumak için kullanılır.

```
data=touch_read_byte(); // Okunan 1 byte bilgi data değişkenine atanıyor.
```

değişken ismi = touch_write_byte(); Fonksiyonu Bu fonksiyon 1-Wire iletişimde slave olan elemana 1 Byte bilgi yazmak için kullanılır. Bu fonksiyon eğer yazma işlemi başarılı bir şekilde gerçekleştiyse "1 (TRUE)" bilgisi ile geri döner. Eğer yazma işlemi sırasında bir sorun çıkmış ise fonksiyon "0 (FALSE)" bilgisi ile geri döner.

```
touch_write_byte(0x33); // 0x33 bilgisi 1-Wire hattındaki cihaza yazılıyor.
```

TOUCH.c dosyasının içeriğinde ilk başta aşağıdaki komutlar vardır. Bu komutlar 1-Wire iletişim için kullanılacak data pin'ini tanımlamak içindir. Normalde bu dosyada 1-Wire iletişim data pin'i RB0 pin'i olarak belirtilmiştir. Bu komutlarda verilen `#define TOUCH_PIN PIN_B0` komutundaki PIN_B0 kısmına 1-Wire iletişiminde data pin'i olarak kullanılması istenen pin ismi yazılarak istenen pin'in data pin'i olarak kullanılması sağlanır.

```
#ifndef TOUCH_PIN
      #define TOUCH_PIN  PIN_B0
#endif
```

Bölüm-21.4'de verilen uygulamayı siz TOUCH.c dosyasını kullanarak programlamaya çalışın. Böylece hem kendi oluşturduğunuz hem de hazır fonksiyonları kullanarak 1-Wire iletişimini gerçekleştirmiş olursunuz. TOUCH.c dosyasını programınıza eklemek için `#include <TOUCH.c>` komutunu programınızda kullanmayı unutmayınız. Ayrıca TOUCH.c dosyası içinde gecikme fonksiyonları kullandığından `#include <TOUCH.c>` komutunu `#USE DELAY (CLOCK=4000000)` komutundan sonra kullanmayı unutmayınız.

22. Bölüm

SICAKLIK SENSÖRÜ, RTC, DİJİTAL POT VE SES UYGULAMALARI

- DS1621 Dijital Sıcaklık Sensörü ve Uygulaması
- DS1302 RTC (Real Time Clock/Calendar) ve Uygulaması
- DS1868 Dijital Potansiyometre ve Uygulaması
- CCS C Derleyicisi Ses Dosyası ve Uygulaması

Bu bölümde DS1621 entegresi ayrıntılı olarak tanıtılmıştır. DS1621 entegresi kontrolü için kendi fonksiyonlarımızın oluşturulması anlatılmış ve bir uygulama gerçekleştirilmiştir. Daha sonra DS1302 RTC (Real Time Clock/Calendar - Gerçek Zaman Saati/Takvimi) entegresi tanıtılmıştır. Bu entegre için CCS C derleyicisi içinde bulunan kütüphane dosyası tanıtılmış ve bir uygulama gerçekleştirilmiştir. Dijital potansiyometre olarak DS1868 entegresi tanıtılmış ve CCS C derleyicisi içinde bulunan kütüphane dosyası ile dijital potansiyometre uygulaması gerçekleştirilmiştir. En son olarak da CCS C derleyicisinde ses ile ilgili kütüphane dosyası tanıtılmış ve bir uygulama gerçekleştirilmiştir.

22.1. DS1621 DİJİTAL SICAKLIK SENSÖRÜ

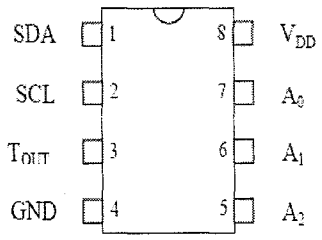
DS1621 I^2C seri iletişim metodunu kullanan sıcaklık ve termostat entegresidir. DS1621'in özelliklerini şöyle sıralayabiliriz;

- Sıcaklık ölçüm sonucunu dijital olarak verir.
- Çalışma gerilimi 2.7V ile 5.5V arasındadır.
- Ölçüm sonucu 9 bit'lik uzunluğundadır.
- I^2C İletişim protokolü ile 2 uçla veri alışverişi sağlanır.
- -55 ^{0}C ile +125 ^{0}C arasında 0.5 ^{0}C çözünürlükte veya Fahrenhayt (Fahrenheit) olarak -67 F ile +257 F arasında 0.9 F çözünürlükte ölçüm yapma kapasitesi vardır.
- 1 saniye'de ölçüm yapıp ve dijital bilgiye çevirebilir.

- Ölçüm için başka bir dış elemana ihtiyaç duymaz.
- 8 pin DIP ve SO paket seçenekleri mevcuttur.

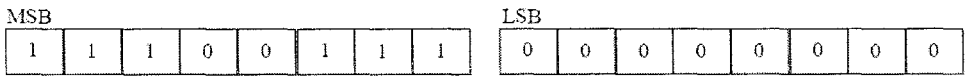
Pin No	Sembol	Açıklama
1	SDA	Seri Veri Giriş/Çıkış
2	SCL	Seri Saat Girişi
3	Tout	Termostat Çıkış Sinyal Ucu
4	GND	Şase
5	A2	Entegre Adres Giriş Ucu
6	A1	Entegre Adres Giriş Ucu
7	A0	Entegre Adres Giriş Ucu
8	Vdd	Pozitif Besleme Ucu

Tablo-22.1. *DS1621 pin açıklamları.*



Şekil-22.1. *DS1621 Pin'leri*

DS1621 entegresi ölçüm sonucunu 9 bit'lik bir değer ile bildirir. Dolayısıyla sonuç değeri için 2 byte'lık veri gönderir. Veri gönderiminde ilk başta yüksek değerlikli kısım (MSB) gönderilir. Ölçüm sonucu değerinin ilk 8 bit'i ölçülen sıcaklık değerinin tam kısmını oluşturur. Diğer byte'ta kalan 1 bit ise ölçüm sonuç değerinin 0.5 ^{0}C ondalıklı olduğu zamanlarda kullanılır. Geriye kalan 7 bit'in değeri ise her zaman 0'dır. Şekil-22.2'de sonuç bit'leri ve Tablo-22.2'de de ölçülen değer ile sıcaklık arasındaki ilişkiye örnek bazı değerler görülmektedir.



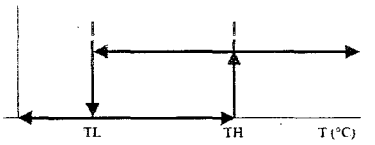
Şekil-22.2. *DS1621 ölçüm sonucu değer formatı.*

Sıcaklık	Dijital Çıkış(Binary Olarak)	Dijital Çıkış(HEX Olarak)
+125°C	01111101 00000000	7B00h
+25°C	00011001 00000000	1900h
+½°C	00000001 00000000	0080h
+0°C	00000000 00000000	0000h
-½°C	11111111 10000000	FF80h
-25°C	11100111 00000000	E700h
-55°C	11001001 00000000	C900h

Tablo-22.2. *DS1621 Sıcaklık-Ölçüm Sonucu İlişki Tablosu.*

DS1621'in termostat özelliği programlanabilir bir histerisiz eğrisi özelliğine sahiptir. Yani komutlarla ölçülen sıcaklık değerinin, istenen bir sıcaklık değeri (**TH**) üstüne çıktığında termostat çıkışının aktif olmasını ve ölçülen sıcaklığın istenen bir sıcaklık değerinden (**TL**) daha az bir sıcaklık değerine gelinceye kadar termostat çıkış sinyalinin aktif kalması sağlanabilir. Termostat çıkışının aktif olması durumu programlanarak değiştirilebilir. Yani termostat aktif olduğunda oluşan çıkış sinyali

ister lojik-0 istenirse lojik-1 olarak ayarlanır. Şekil-22.3'de aktif durumu lojik-1 olarak ayarlanan bir DS1621 entegresinin termostat histerisiz eğrisi görülmektedir.



Şekil-22.3. *Aktifken Lojik-1 seviyesinde olan DS1621 termostat histeristiği.*

DS1621 içinde bir konfigürasyon/durum kaydedicisi içerir. Bu kaydedici bit'lerinin durumlarına göre çeşitli ayarlamalar yapıldığı gibi bazı işlemlerin durumları da bu kaydedici bit'leri ile öğrenilebilir. Şekil-22.4'de bu kaydedici görülmektedir.

DONE	THF	TLF	NVB	1	0	POL	1SHOT

Şekil-22.4. *DS1621 Konfigürasyon/Durum kaydedicisi.*

DONE : Sıcaklık dönüştürme işlemi tamamlandı bit'i. Bu bit "1" ise dönüştürme tamamlandı, "0" ise dönüştürme işlemi devam ediyor demek.

THF : **T**emperature **H**igh **F**lag. Termostat sıcaklığının üst değer bayrak bit'i. Termostat için belirlenen üst sıcaklığa gelindiğinde veya bu sıcaklık aşıldığında bu bit "1" olur. Bu durumda lojik-1 olan bu bit silinene kadar bu durumunu korur. Bu bit "0" yapılarak bu durumdan çıkartılabilir.

TLF : **T**emperature **L**ow **F**lag. Termostat sıcaklığının alt değer bayrak bit'i. Termostat için belirlenen alt sıcaklığa gelindiğinde veya bu sıcaklıktan daha aşağıya gelindiğinde bu bit "1" olur. Bu durumda lojik-1 olan bu bit silinene kadar bu durumunu korur. Bu bit "0" yapılarak sistem bu durumdan çıkartılabilir.

NVB : **N**on**v**olatile Memory **B**usy Flag. Kalıcı hafıza meşgul bayrağı. Bu bit "1" ise E^2 hafızası işlem görmekte yani meşguldür. Eğer bu bit "0" ise kalıcı hafıza meşgul değildir.

POL : Output **Pol**arity Bit. Termostat için belirlenen alt ve üst sıcaklık set değerleri sınırları aşıldığında verilen alarm sinyalinin lojik-0 veya lojik-1 olacağını belirleyen bit'tir. Bu bit "1" ise alarm oluştuğunda çıkış sinyali lojik-1, bu bit "0" ise alarm oluştuğunda çıkış sinyali lojik-0 olur.

1SHOT : One **Shot** Mode. Bu bit "1" ise Start Convert yani sıcaklık ölçme komutu gelince bir kez ölçme yapar ve durur. Diğer sıcaklık ölçme komutu gelene kadar başka ölçme yapmaz. Bu bit "0" olduğunda ise ölçme işlemi devamlı olarak yapılır.

Önceden de belirtildiği gibi DS1621 ile I^2C protokolü üzerinden iletişim kurulabilir. DS1621 standart I^2C iletişim modunda 100Khz, hızlı (fast) modda ise 400Khz hızında çalışabilir. Bu iletişimde mikrodenetleyici master konumunda, DS1621 ise slave (uydu) konumundadır. Yani iletişimde saat sinyali mikrodenetleyici tarafından gönderilecektir. Bu iletişim belli kurallar çerçevesinde yapılır.

SDA hattı meşgul olmadığı zamanlar veri gönderilmeli
SCL saat sinyali hattı "lojik-1" iken veri hattının seviyesi aynı kalmalıdır. Aksi takdir de oluşan değişiklikler kontrol sinyali olarak algılanır.

DS 1621 entegresinin 8 bit'lik adres bilgisinin ilk 4 bit'i cihaz kontrol kodudur. Buda DS 1621 için 1001 (0x9X) dur. Diğer 3 bit ise (A2, A1, A0) cihaz seçme bit'idir. 8. bit ise R/W bit'idir. Bu bit ile de "1" ise okuma "0" ise yazma işlemi olacağı belirtilir.

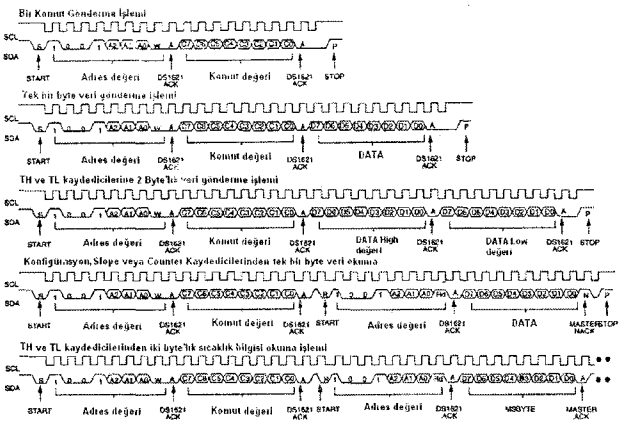
DS 1621 entegresini kontrol komutları vardır. Bu komutlar I^2C protokolünde belirtilen yazma işlemi ile master olan cihazdan (denetleyiciden) DS1621'e yani slave konumda olan cihaza gönderilir. Komutlar 8 bit'liktir. Tablo-22.3'de bu komutlar ve açıklamaları görülmektedir. DS1621'in TH, TL ve konfigürasyon kaydedicilerine yazma işlemi oda sıcaklığındaki ortamlar için yaklaşık 10 ms sürmektedir. Bu nedenle yazma komutundan sonra en azından 10 ms'lik bir sürede herhangi bir işlem yapılmamalıdır.

KOMUT	AÇIKLAMA	DEĞER	Komut Gönderildikten Sonra Yapılacak İşlem
SICAKLIK ÖLÇME KOMUTLARI			
Sıcaklığı Oku	Ölçümü yapılan son sıcaklık değerini kaydediciden okur.	AAh	2 byte bilgi okunur.
Sayıcıyı Oku	Sayıcının değerini okur. R/W bit'i 1 olduğunda geçerlidir.	A8h	1 byte bilgi okunur.
Slope Sayıcıyı Oku	Slope sayıcısının değerini okur.R/W bit'i 1 olduğunda geçerlidir.	A9h	1 byte bilgi okunur.
Sıcaklık Ölçmeye Başla	Sıcaklık ölçmeyi başlatır.	EEh	Başka işleme gerek yok.
Sıcaklık Ölçmeyi Durdur	Sıcaklık ölçmeyi durdurur.	22h	Başka işleme gerek yok.
TERMOSTAT KOMUTLARI			
TH'a Erişmek	TH kaydedicisine Okuma veya Yazma işlemini yapar. R/W bit'i "0" ise TH kaydedicisine bilgi yazılır. Bu komuttan sonraki gönderilen 2 byte TH'a yazılacak	A1h	2 byte bilgi yazılır. 2 byte bilgi okunur.

	veridir. R/W bit'i "1" ise TH'ın içeriği aynı şekilde okunur.		
TL'ye Erişmek	TL kaydedicisine Okuma veya Yazma işlemini yapar. R/W bit'i "0" ise TL kaydedicisine bilgi yazılır. Bu komuttan sonraki gönderilen 2 byte TL'a yazılacak veridir. R/W bit'i "1" ise TL'ın içeriği aynı şekilde okunur.	A2h	2 byte bilgi yazılır. 2 byte bilgi okunur.
Konfigürasyon Kaydedicisine Erişim	Konfigürasyon kaydedicisine bilgi yazma veya okuma işlemi yapar. R/W bit'i "1" ise okuma, "0" ise yazma işlemi yapılır. Okuma veya yazma işlemlerinde 1 byte'lık bilgi okunur veya yazılır.	ACh	1 byte bilgi yazılır. 1 byte bilgi okunur.

Tablo-22.3. *DS1621 Komutları ve açıklamaları.*

Tablo-19.3'de verilen komutların okuma ve yazma işlemlerinde I^2C protokolünde nasıl kullanılacağına ilişkin bilgiler aşağıda verilen Şekil-22.5'deki diyagramda görülmektedir. Yapılacak işlemler ve okuma-yazma sırasındaki protokol işlemleri net bir şekilde diyagramdan görülebilir.



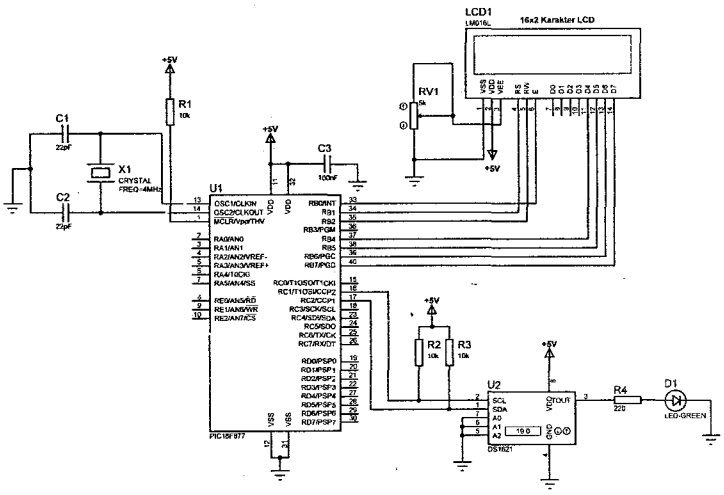
Şekil-22.5. *DS1621 okuma ve yazma için I^2C protokolü işlemleri.*

22.1.1. DS1621 Sıcaklık ve Termostat Entegresi Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada DS1621 entegresinin hem sıcaklık ölçümü hem de termostat özelliği kullanılmıştır. DS1621'in termostat alarm çıkışına LED bağlanmıştır. Böylece belirlenen sıcaklık aralığının dışında alarm sinyali gelince LED yanacaktır. Devre programında DS1621 için CCS C kütüphanesinde bulunan hazır DS1621.c dosyası **kullanılmamıştır.** Bunun yerine I^2C iletişimi daha iyi anlaşılsın diye **DS1621**

entegresi kontrol fonksiyonları kendimiz tarafından yazılmıştır. İstenirse C:\Program Files\PICC\Drivers\ dizininde bulunan ds1621.c dosyası da kullanılabilirdi.



Şekil-22.6. *DS1621 Sıcaklık ve Termostat entegresi uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 PIC16F877 ile DS1621 Sıcaklık ve Termostat Entegresi Uygulaması
*****************************************************************/
#include <16f877.h>          // Kullanılacak denetleyicinin başlık
                             // dosyası tanıtılıyor.
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
// Denetleyici konfigürasyon ayarları
#use delay (clock=4000000)  // Gecikme fonksiyonu için kullanılacak
                             // osilatör frekansı belirtiliyor.
#define use_portb_lcd TRUE   // LCD B port'una bağlı
#include <lcd.c>

#use i2c(master,sda=pin_c1, scl=pin_c0, slow=100000)

//************ DS1621 SICAKLIK OKUMA FONKSİYONU *************
float ds1621_oku (void)
{
   float th,tl;
   float sicaklik;

   i2c_start();     // i2c iletişimi başlangıç ayarları
   i2c_write(0x90);// Adres ve R/W bit bilgisi gönderiliyor
                    // R/W bit'i yazma yapılacağı için "0"
   i2c_write(0x22);// sıcaklık dönüşümünü durdur komutu gönderiliyor
   i2c_stop();      // i2c iletişimini durdur
```

```
   i2c_start();      // i2c iletişimi başlangıç ayarları
   i2c_write(0x90); // Adres ve R/W bit bilgisi gönderiliyor
                    // yazma yapılacağından burada R/W bit'i "0"
   i2c_write(0xee); // Sıcaklık dönüşümüne başla komutu gönderiliyor
   i2c_stop();

   i2c_start();
   i2c_write(0x90);
   i2c_write(0xaa);  // Sıcaklığı oku komutu gönderiliyor
   i2c_start();
   i2c_write(0x91);  // Adres ve R/W bit bilgisi gönderiliyor
                     // okuma yapılacağından burada R/W bit'i "1"
   th=i2c_read(0);   // sıcaklık değerinin yüksek 8 bit'lik değerini
                      // oku ve değişkene kaydet
   i2c_start();
   i2c_write(0x91);
   tl=i2c_read(0);    // sıcaklık değerinin düşük 8 bit'lik değerini
                      // oku ve değişkene kaydet
   i2c_stop();

   if (th>0x80)
      th=th-(0xFF+1);

   if (tl==0x80)//Eğer sıcaklık değeri buçukluysa sıcaklığa 0.5 ekle
      sicaklik=th+0.5;
   else
      sicaklik=th; // Eğer sıcaklık değeri buçuklu değilse aynen
                   // değişkene kaydet

   delay_ms(50);
   return(sicaklik);    // Fonksiyon "sicaklik" değeri ile geri döner
}

//************ DS1621 TH'a VERİ YAZMA FONKSİYONU ********************
void yaz_ds1621_th_tl (byte th_h, byte th_l, byte tl_h, byte tl_l)
{
   i2c_start();        // i2c iletişimi başlangıç ayarları
   i2c_write(0x90);    // Adres ve R/W bit bilgisi gönderiliyor
                       // yazma yapılacağından burada R/W bit'i "0"
   i2c_write(0xa1);    // TH'a erişme komutu gönderiliyor
   i2c_write(th_h);    // TH değerinin yüksek değerlikli byte'ı
                       // gönderiliyor
   i2c_write(th_l);    // TH değerinin düşük değerlikli byte'ı
                            // gönderiliyor
   i2c_stop();         // i2c iletişimi sonlandırılıyor
   delay_ms(11);       // Belli bir süre bekleniyor

   i2c_start();
   i2c_write(0x90);
   i2c_write(0xa2);     // TL'ye erişme komutu gönderiliyor.
   i2c_write(tl_h);     // TL değerinin yüksek değerlikli byte'ı
                        // gönderiliyor.
```

```
   i2c_write(tl_l);     // TL değerinin düşük değerlikli byte'ı
                        // gönderiliyor
   i2c_stop();
   delay_ms(11);
}

//** DS1621 KONFİGÜRASYON KAYDEDİCİSİNE VERİ YAZMA FONKSİYONU ***
void yaz_ds1621_ayar (byte deger)
{
   i2c_start();
   i2c_write(0x90);
   i2c_write(0xac);   // Konfigürasyon kaydedicisine erişme komutu
                      // gönderiliyor.
   i2c_write(deger);  // Konfigürasyon ayar bilgisi gönderiliyor.
   i2c_stop();
   delay_ms(11);
}

//******* DS1621 KONFİGÜRASYON KAYDEDİCİSİNİ OKUMA FONKSİYONU *****
byte oku_ds1621_ayar ()
{
   byte con;

   i2c_start();
   i2c_write(0x90);  // Adres ve R/W bit bilgisi gönderiliyor.
                     // yazma yapılacağından burada R/W bit'i "0"
   i2c_write(0xac);   // Konfigürasyon kaydedicisine erişme komutu
                      // gönderiliyor.
   i2c_start();
   i2c_write(0x91);  // Adres ve R/W bit bilgisi gönderiliyor.
                     // okuma yapılacağından burada R/W bit'i "1"
   con=i2c_read(0);  // Konfigürasyon kaydedicisi içeriği okunuyor.
   i2c_stop();
   return(con);  // Fonksiyon "con" değişkeni değeri ile geri döner.
}

char sonuc[17];
byte ayar;

//******************* ANA PROGRAM FONKSİYONU *******************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   lcd_init(); // LCD başlangıç ayarları yapılıyor
```

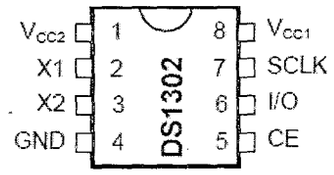
```
   delay_ms(5);

   ayar=oku_ds1621_ayar();  // DS1621 konfigürasyon kaydedicisi
                            // içeriği okunuyor.
   bit_set(ayar,1);//Konfigürasyon kaydedicisinde POL bitini "1" yap
   // bu bit alarm anında çıkışın lojik-1 olması için "1" yapılıyor
   bit_set(ayar,0);      // Konfigürasyon kaydedicisinde 1SHOT bit'i
       // "1" yapılıyor sürekli sıcaklık okuması yapılsın diye
   yaz_ds1621_ayar(ayar);     // Yapılan değişiklikler konfigürasyon
                              // kaydedicisine aktarılıyor
   yaz_ds1621_th_tl(22,0,21,5);  // TH ve TL değerleri belirleniyor

   while(1)
   {
      sprintf(sonuc,"\fSicaklik=%0.1fC",ds1621_oku());
      printf(lcd_putc,"%s",sonuc);
      delay_ms(500);
   }
}
//*****************************************************************
```

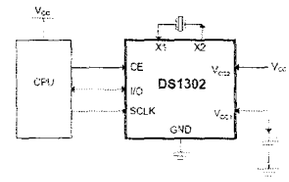
Devrenin programında görüleceği üzere DS1621 için çeşitli fonksiyonlar tanımlanmıştır. Bu fonksiyonlar tek bir dosyada toplanıp programın başında header (başlık) dosyası olarak da tanımlanabilirdi. Program ile DS1621'in sürekli sıcaklığı okuması sağlanmış ve bu sıcaklık LCD ekranda görüntülenmiştir. Aynı zamanda DS1621 termostat değerleri olarak 21.5 ^{0}C ile 22 ^{0}C arası belirlenmiştir. Böylece ortam sıcaklığı 22 ^{0}C'yi aşınca alarm verilecek ve termostat çıkışı lojik-1 olacaktır. Dolayısıyla LED yanacaktır. Sıcaklık 21.5 ^{0}C düşene kadar alarm konumu devam edecektir. Alarm durumunda termostat çıkışı lojik-1 olarak ayarlanmıştır. Programda yapılan değişiklikle alarm çıkışı lojik-0'da yapılabilir.

22.2. DS1302 RTC (REAL TIME CLOCK/CALENDAR)

DS1302 entegresi, RTC (Real Time Clock/Calendar - Gerçek Zaman Saati/Takvimi) olarak adlandırılan bir saniye, dakika, saat, gün, ay ve yıl sayacı ve saklayıcısıdır. Entegre içinde 31 Byte'lık bir statik RAM hafıza mevcuttur. DS1302 2100 yılına kadar takvim bilgilerini içermektedir. DS1302 entegresine bağlanan harici bir pil ile entegrenin beslemesi kesilse dahi entegre zaman ve takvim sayma işlemlerine devam edebilmektedir. DS1302 2V ile 5.5V arasında istenen değerde çalışabilir. 2V ile beslendiğinde 300nA'den daha az akım çekerek pil ile bile uzun süre kullanılabilir. **DS1302 3-Wire iletişim protokolü ile kontrol edilmektedir.** DS1302 entegresine, saat ve takvim sayma işlemleri için **X1** ve **X2** pin'lerine harici bir **32.768Khz'lik kristal** bağlanır. İstenirse kristal yerine X1 pin'ine 32.768Khz'lik osilatör devresi de bağlanabilir. DS1302 pin'lerinde $\mathbf{V_{CC1}}$ ucu pil beslemesinin, $\mathbf{V_{CC2}}$ ucu ise normal besleme kaynağının ucudur. Eğer V_{CC2} gerilimi V_{CC1}+0.2V değerinden büyük ise DS1302 V_{CC2} ucundan besleme gerilimini alır. Eğer V_{CC2} gerilimi V_{CC1} geriliminden küçük ise DS1302 V_{CC1} ucundan besleme gerilimini alır. **CE**

ucu DS1302 entegresinde okuma ve yazma işlemleri sırasında lojik-1 seviyesinde olmalıdır. DS1302'nin önceki teknik bilgi notlarında bu uç **RST** olarak belirtilmiştir. **SCLK** pin'i clock (saat) sinyali giriş ucu, **I/O** pin'i ise veri giriş/çıkış ucudur.





Şekil-22.7. *DS1302 PDIP kılıf paketi* ***Şekil-22.8.*** *DS1302 tipik uygulama devresi*

DS1302 gün bilgilerini tam sayı olarak algılar ve tam sayı olarak verir. Bu sayılar 1, 2, 3, 4, 5, 6, 7 sayılarıdır. Bu sayılar her yeni günde sırayla artar. Programcı bu sayılara istediği değeri yükleyebilir. Örneğin "1" sayısına Pazar, "2" sayısına Pazartesi, "3" sayısına Salı gibi.

22.2.1. CCS C Derleyicisi DS1302 Kütüphane Dosyası

DS1302 entegresi için DS1621 entegresi gibi kendi fonksiyonlarımızı oluşturmadık. CCS C derleyicisinde bu iş için C:\Program Files\PICC\Driver\ dizi içinde **"DS1302.c"** dosyası bulunmaktadır. İsterseniz DS1302'nin teknik bilgi notlarına bakarak kendi fonksiyonlarınızı da bu bölüme kadar edindiğiniz bilgi ve tecrübe ışığında oluşturabilirsiniz. Aşağıda DS1302.c dosyası içinde tanımlanmış fonksiyonlar ve görevleri verilmiştir.

rtc_init(); Fonksiyonu = Bu fonksiyon diğer fonksiyonlar çağrılmadan önce mutlaka çağrılmalıdır.

rtc_set_datetime(*gün,ay,yıl,haftanın günü,saat,dakika*); Fonksiyonu = Bu fonksiyon ile gün, ay, yıl, haftanın günü, saat ve dakika bilgileri ayarlanır.

rtc_get_date(*gün,ay,yıl,haftanın günü*); Fonksiyonu = Bu fonksiyon DS1302'den gün, ay, yıl ve haftanın günü bilgilerini almaya yarar.

rtc_get_time(*saat,dakika,saniye*); Fonksiyonu = Bu fonksiyon DS1302'den saat, dakika ve saniye bilgilerini almaya yarar.

rtc_write_nvr(*adres, data*); Fonksiyonu = Bu fonksiyon ile DS1302 entegresinin RAM hafızasında istenen adrese istenen bilgi yazılır.

değişken ismi = **rtc_read_nvr(*adres*); Fonksiyonu** = Bu fonksiyon ile DS1302 entegresinin RAM hafızasındaki istenen bir adresteki bilgi okunur.

get_bcd(*data*); Fonksiyonu = Bu fonksiyon data değişkeni içeriğini BCD formatına çevirir.

rm_bcd(*data*); Fonksiyonu = Bu fonksiyon data değişkeni içeriğini tam sayı formatına çevirir.

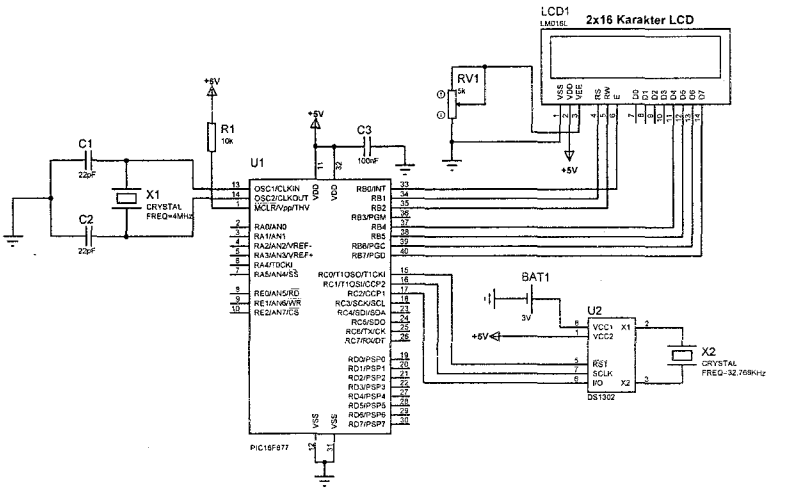
"DS1302.c" dosyası içinde en başta aşağıda verilen komutlar vardır. Bu komutlar ile DS1302'nin 3-Wire iletişim protokolünde kullanacağı pin'ler belirtilmektedir. Bu komutlardaki pin isimleri değiştirilerek iletişim için istenen pin'lerin kullanılması sağlanır.

```
#ifndef RTC_SCLK
        #define RTC_SCLK PIN_B1 // SCLK pin'i tanımlanıyor
        #define RTC_IO   PIN_B3 // I/O pin'i tanımlanıyor
        #define RTC_RST  PIN_B2 // CE (RST) pin'i tanımlanıyor
#endif
```

22.2.2. DS1302 RTC Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada DS1302 entegresi ve LCD ile bir devre gerçekleştirilmiştir. LCD ekranda ilk satırda tarih ve o günkü gün bilgisi, ikinci satırda saat, dakika, saniye bilgileri gösterilmiştir.



Şekil-22.9. *DS1302 RTC uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
 PIC16F877 ile DS1621 Sıcaklık ve Termostat Entegresi Uygulaması
*****************************************************************/
#include   <16f877.h>      //  Kullanılacak  denetleyicinin  başlık
                           // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                           // osilatör frekansı belirtiliyor.
```

```
#define use_portb_lcd TRUE    // LCD B port'una bağlı
#include <lcd.c>      // LCD.c dosyası programa ekleniyor
#include <DS1302.c>   // DS1302.c dosyası programa ekleniyor

// byte türünde değişkenler tanımlanıyor
byte saniye,dakika,saat,gun,ay,yil,haftanin_gunu;

//******************** ANA PROGRAM FONKSİYONU ********************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   lcd_init();  // LCD başlangıç ayarları yapılıyor
   rtc_init();  // DS1302 başlangıç ayarları yapılıyor

   rtc_set_datetime(25,07,07,4,22,52); //Tarih,Saat ayarı yapılıyor

   while(1)
   {
      rtc_get_time(saat,dakika,saniye); // DS1302'den saat,
                                        // dakika,saniye bilgileri alınıyor
      rtc_get_date(gun,ay,yil,haftanin_gunu); // DS1302'den gün,ay,
                                        // yil ve günün isim bilgisi alınıyor
      printf(lcd_putc,"\fT:%02d/%02d/20%02d ",gun,ay,yil); //LCD'ye
                                        // gün,ay,yıl bilgileri yazdırılıyor

      switch(haftanin_gunu) // DS1302'den sayısal olarak alınan
       // haftanın günü bilgisine metin olarak karşılık atanıyor
      {
         case 1: printf(lcd_putc,"PAZ"); break; // Eğer "1" ise
                                                // LCD'ye PAZ yaz
         case 2: printf(lcd_putc,"PZT"); break; // Eğer "2" ise
                                                // LCD'ye PZT yaz
         case 3: printf(lcd_putc,"SAL"); break; // Eğer "3" ise
                                                // LCD'ye SAL yaz
         case 4: printf(lcd_putc,"CRS"); break; // Eğer "4" ise
                                                // LCD'ye CRS yaz
         case 5: printf(lcd_putc,"PER"); break; // Eğer "5" ise
                                                // LCD'ye PER yaz
         case 6: printf(lcd_putc,"CUM"); break; // Eğer "6" ise
                                                // LCD'ye CUM yaz
         case 7: printf(lcd_putc,"CTS"); break; // Eğer "7" ise
                                                // LCD'ye CTS yaz
      }
```

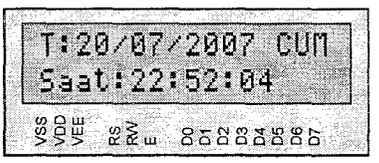
```
        printf(lcd_putc,"\nSaat:%02d:%02d:%02d",saat,dakika,saniye);
        // LCD'ye saat,dakika ve saniye değerlerini yaz
        delay_ms(1000); // 1 sn gecikme veriliyor
    }
}
//*****************************************************************
```

Programda `#include <DS1302.c>` komutu ile DS1302 kontrol fonksiyonlarının olduğu dosya programa eklenmiştir. `byte saniye,dakika,saat,gun,ay,yil,haftanin_gunu;` komutu ile de fonksiyonlar da kullanacağımız byte türünde değişkenler tanımlanmıştır. Ana fonksiyonda `rtc_init();` komutu ile DS1302 entegresi başlangıç ayarları yapılmıştır. Ardından `rtc_set_datetime(20,07,07,4,22,52);` komutu ile DS1302'ye enerji verildiğinde saymaya başlayacağı sırayla, gün, ay, yıl, haftanın günü, saat ve dakika değerleri belirtilmiştir. Enerji geldiğinde DS1302 bu başlangıç değerlerinden itibaren saymaya başlar. Bu komutla programımıza göre başlangıç ayarları 20/07/07 Cuma saat 22:52 olarak belirtilmiştir.

Sonsuz döngüde `rtc_get_time(saat,dakika,saniye);` komutu ile güncel saat, dakika ve saniye bilgileri DS1302'den alınmaktadır. Ardından `rtc_get_date(gun,ay,yil,haftanin_gunu);` komutu ile DS1302'den güncel gün, ay, yıl ve haftanın günü bilgileri alınmaktadır.

`printf(lcd_putc,"\fT:%02d/%02d/20%02d ",gun,ay,yil);` komutu ile LCD ekran temizlenmiş ve 1.satır, 1.sütuna ilk başta "T:" karakterleri yazdırılmıştır. Ardından "gun", "ay" değişkenleri değerleri 2 basamaklı olarak (02% komutu sayesinde) LCD'ye yazdırılmıştır. Bu değerlerin arasına ise / karakteri konulmuştur. Daha sonra LCD'ye "20" sayısı yazdırılmıştır. Ardından "yil" değişkeni 2 basamaklı olarak LCD'ye yazdırılmıştır. Böylece yıl değeri 2007 şeklinde görülebilecektir.

Haftanın günlerini DS1302 önceden söylendiği gibi sayısal olarak (1, 2, 3, 4, 5, 6, 7 ile) verir. Bu nedenle bu değerlere biz gün isimlerini yüklememiz gerekir. Bu iş için switch-case döngüsü kullanılmıştır. `switch(haftanin_gunu)` komutu ile haftanin_gunu değişkeni içeriği denetlenmektedir. `case 1: printf(lcd_putc,"PAZ"); break;` komutlarıyla eğer DS1302'den gelen haftanın günü bilgisi "1" ise LCD'ye "PAZ" yazılır. `case 2: printf(lcd_putc,"PZT"); break;` komutlarıyla eğer DS1302'den gelen haftanın günü bilgisi "2" ise LCD'ye "SAL" yazılır. Diğer switch-case komutları da aynı mantık ile çalışır. En son olarak da `delay_ms(1000);` komutu ile 1 sn gecikme verilmiştir.

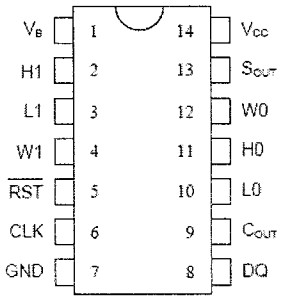


Şekil-22.10. *DS1302 uygulaması LCD görüntüsü.*

22.3. DS1868 DİJİTAL POTANSİYOMETRE ENTEGRESİ

Dijital potansiyometreler analog potansiyometrelerin yaptığı işi sayısal veriler ile gerçekleştiren elemanlardır. DS1868 entegresi Maxim-Dallas firmasının bir dijital potansiyometre ürünüdür. **DS1868 içinde iki adet (0. ve 1. potansiyometre) dijital olarak kontrol edilebilen 256 değişik pozisyonda ayarlanabilen potansiyometre içerir.** DS1868 çok düşük güç tüketimine ve sessiz çalışmaya sahiptir. 5V veya 3V gerilim ile çalışabilirler. DS1868 entegresi 3-Wire iletişim protokolü ile kontrol edilmektedir. DS1868 serisinin 3 türü vardır. Bu 3 tür değişik direnç değerlerine sahiptir.

DS1868-10 = Yaklaşık 10KΩ
DS1868-50 = Yaklaşık 50KΩ
DS1868-150 = Yaklaşık 150KΩ

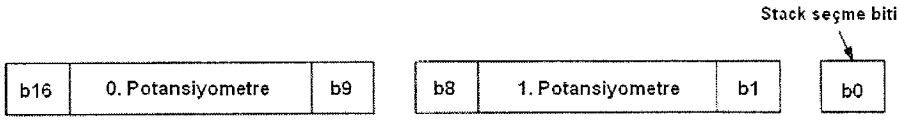


Şekil-22.11. *DS1868 pin diyagramı.*

Pin No	Pin İsmi	Pin Görevi
1	Vb	Alt tabaka ön gerilimi
2	H1	1.Potansiyometrenin üst ucu
3	L1	1.Potansiyometrenin alt ucu
4	W1	1.Potansiyometre orta ucu
5	RST	Seri port reset girişi
6	CLK	Seri port Clock (saat) sinyali girişi
7	GND	Şase
8	DQ	Seri port data girişi
9	Cout	Kaskat port çıkışı
10	L0	0.Potansiyometrenin alt ucu
11	H0	0.Potansiyometrenin üst ucu
12	W0	0.Potansiyometre orta ucu
13	Sout	0. ve 1. potansiyometreler seri bağlandığında meydana gelen potansiyometre ortak uç çıkışı
14	Vcc	+5V besleme ucu

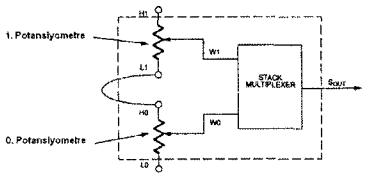
Tablo-22.4. *DS1868 pin açıklamaları.*

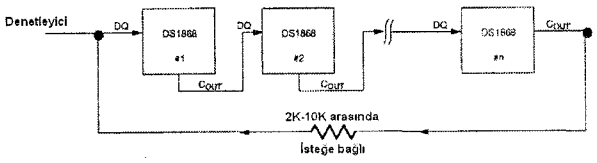
DS1868 içinde 0. ve 1. potansiyometre değerlerinin kaydedildiği kaymalı kaydedici yapısında iki adet stack kaydedici vardır. Bu kaydediciler toplam 17 bit'ten oluşmaktadır. Düşük değerlikli ilk bit stack kaydedicisini seçme bit'idir. 1-8. bit'ler 1. potansiyometre için, 9-16. bit'ler 0. potansiyometre içindir. Denetleyicinin DQ ucundan gönderdiği bilgiler sırayla bu kaydedicilerde saklanır. Bu değerler potansiyometrelerin direnç değerini oluşturur.



Şekil-22.12. *17 bit Giriş/Çıkış (I/O) kaymalı kaydedici.*

DS1868 içinde bulunan 2 adet potansiyometre birbirine seri olarak bağlanabilir. Bu durum Şekil-22.13'de görülmektedir. İstenirse birden fazla DS1868 ard arda birbirine bağlanabilir. Bu durumda ilk DS1868'in C_{OUT} çıkışı diğer DS1868'in DQ girişine bağlanır. Şekil-22.14'de bu durum görülmektedir. Şekilde görüldüğü gibi istenirse ard arda bağlantıda son DS1868 elemanının C_{OUT} çıkışı ilk DS1868 elemanın DQ girişine seri bir direnç üzerinden (2K ile 10K arasında bir değerde) verilebilir. Bu durum bize gönderilen bilginin doğru olarak yazılıp yazılmadığını denetleme imkânı verir.





Şekil-22.13. *DS1868 potansiyometrelerinin seri bağlanması.*

Şekil-22.14. *Ard arda DS1868 bağlantısı.*

22.3.1. CCS C Derleyicisi DS1868 Kütüphane Dosyası

CCS C derleyicisinde DS1868 dijital pot entegresinin kontrolü için C:\Program Files\PICC\Driver\ dizi içinde **"DS1868.c"** dosyası bulunmaktadır. İsterseniz kendi fonksiyonlarınızı da bu bölüme kadar edindiğiniz bilgi ve tecrübe ışığında oluşturabilirsiniz. Aşağıda DS1868.c dosyası içinde tanımlanmış fonksiyonlar ve görevleri verilmiştir.

init_pots(); Fonksiyonu = Tüm potansiyometre değerlerini 0 (sıfır) yapar.

set_pot(***potansiyometre numarası, potansiyometre direnç değeri*****); Fonksiyonu =** Bu fonksiyon ile istenen potansiyometre değeri istenen direnç değerine getirilir. "Potansiyometre numarası" kısmına 0 veya 1 değerleri yazılabilir. "Potansiyometre direnç değeri" kısmına da tam sayı değerleri yazılır. Örneğin DS1868-100 entegresi 100KΩ direnç değerine sahiptir. **DS1868 entegresi bu direnç değerini 256 pozisyon değerine böler. Dolayısıyla 100KΩ/256=100000Ω/256=390.625Ω değeri elde edilir. Yani bu fonksiyonda yazdığınız her değer 390.625Ω ile çarpılmakta ve çıkış direnç değeri böyle elde edilmektedir.**

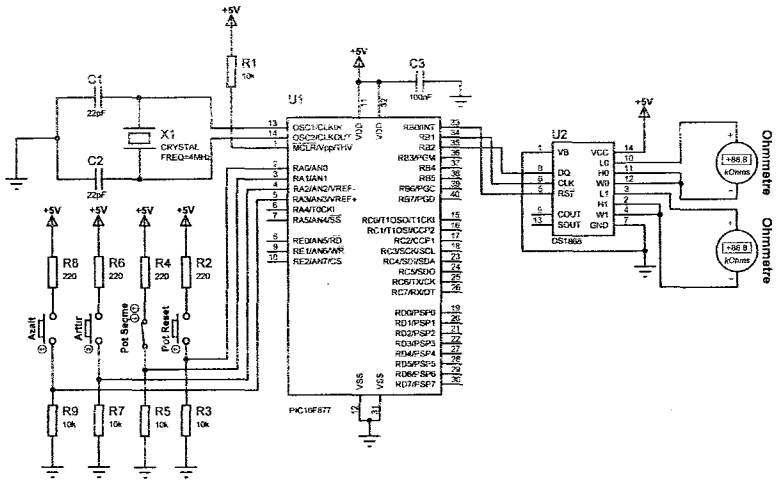
"DS1868.c" dosyası içinde en başta aşağıda verilen komutlar vardır. Bu komutlar ile DS1868'in iletişim esnasında hangi pin'leri kullanacağı belirtilmektedir. Bu komutlardaki pin isimleri değiştirilerek iletişim için istenen pin'lerin kullanılması sağlanır.

```
#define RST1 PIN_B0   // RST ucu RB0 pin'ine bağlı
#define CLK  PIN_B1   // CLK ucu RB1 pin'ine bağlı
#define DI   PIN_B2   // DI (DQ) ucu RB2 pin'ine bağlı
#define NUM_POTS 2    // Pot sayısı (NUM_POTS) 2 olarak belirtiliyor
```

22.3.2. DS1868 Dijital Potansiyometre Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Uygulamada DS1868-100 dijital pot entegresi kullanılmıştır. Bu entegre yaklaşık 100KΩ değerinde 2 adet potansiyometreye sahiptir. Dolayısıyla bu değer 256 basamak olarak sunulabildiğinden, her basamak değeri 100KΩ/256=100000Ω/256=390.625Ω olarak elde edilir. Devrede 3 adet buton ve 1 adet anahtar bulunmaktadır. Bu butonlar ve anahtar pot seçme, seçilen pot değerini arttırma ve azaltma ile 2 port değerini de sıfırlama amaçları için kullanılmaktadır. Pot seçme anahtarı kapalı iken 0. potansiyometre, açık iken 1. potansiyometre seçilmiş demektir.



Şekil-22.15. *DS1868 Dijital Potansiyometre uygulaması devre şeması.*

```
/*****************************************************************
 PIC16F877 ile DS1868 Dijital Potansiyometre Uygulaması
*****************************************************************/
#include <16f877.h>          // Kullanılacak denetleyicinin başlık
                             // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000) // Gecikme fonksiyonu için kullanılacak
                             // osilatör frekansı belirtiliyor.
#include <DS1868.c>  // DS1868.c dosyası programa ekleniyor

// Butonlar ve Anahtar Tanımlamaları Yapılıyor
#define pot_reset pin_a0
#define pot_sec   pin_a1
#define arttir    pin_a2
#define azalt     pin_a3
int i=0,j=0; // Tam sayı tipinde değişkenler tanımlanıyor
```

```
//******************* ANA PROGRAM FONKSİYONU *******************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   init_pots(); // 2 potansiyometre değeri ilk anda sıfırlanıyor

   while(1) // Sonsuz döngü
   {
     if (input(pot_reset))   // Eğer Pot reset butonuna basıldı ise
     {
        init_pots();         // 2 potansiyometreyi de sıfırla
        i=0; j=0;            // i ve j değerleri de sıfırla
     }
     if (input(pot_sec))     // 0. Potansiyometre seçilmiş ise
     {
        if (input(arttir))   // Arttir butonuna basılmış ise
        {
           i++;              // i değerini bir arttır
           set_pot(0,i);     // 0. potansiyometreyi i değerine ayarla
           while(input(arttir)); // Butondan el çekile kadar bekle
        }
        if (input(azalt) & (i!=0)) // Arttir butonuna basılmış ve
        {                          // i değeri 0 değil ise
           i--;                    // i değerini bir azalt
           set_pot(0,i);     // 0. potansiyometreyi i değerine ayarla
           while(input(azalt));   // Butondan el çekile kadar bekle
        }
     }
     else  // 1. potansiyometre seçilmiş ise
     {
        if (input(arttir))        // Arttir butonuna basılmış ise
        {
           j++;                   // j değerini bir arttır
           set_pot(1,j);     // 1. potansiyometreyi j değerine ayarla
           while(input(arttir)); // Butondan el çekile kadar bekle
        }
        if (input(azalt) & (j!=0)) // Arttir butonuna basılmış ise
        {
           j--;              // i değerini bir azalt
           set_pot(1,j);     // 1. potansiyometreyi j değerine ayarla
           while(input(azalt)); // Butondan el çekile kadar bekle
        }
     }
   }
```

```
}
/*****************************************************************
```

Programda `#include <DS1868.c>` komutu ile DS1868 entegresi kontrol fonksiyonları için CCS C'de bulunan dosya programa eklenmiştir. Ana fonksiyonda `init_pots();` komutu ile ilk başta DS1868'in 2 potansiyometresi de sıfırlanmıştır. Sonsuz döngüde `if (input(pot_reset))` komutu ile `pot_reset` butonuna basılıp basılmadığı denetlenmiştir. Eğer bu butona basılmış ise `init_pots();` komutu aktif olacak ve DS1868'in 2 potansiyometresi de sıfırlanacaktır. Ayrıca i=0 ve j=0 komutları ile bu değişken içerikleri de sıfırlanmıştır.

`if (input(pot_sec))` komutu ile `pot_sec` anahtarının kapalı olup olmadığı kontrol edilmektedir. Eğer `pot_sec` anahtarı kapalı ise 0. potansiyometre ayarları için bu döngü aktif olacaktır. Bu döngü içinde `if (input(arttir))` komutu ile `arttir` butonuna basılıp basılmadığı kontrol edilmektedir. Bu butona basılmış ise i değişkeni değeri `i++;` komutuyla bir arttırılarak `set_pot(0,i);` komutu ile de 0.potansiyometre değeri belirlenmektedir. `while(input(arttir));` komutu ile de butondan el çekilene kadar `while` döngüsünün işlemesi dolayısıyla butona her basışta sadece 1 arttırma işlemi yapılması sağlanmıştır. Yine aynı döngüde `if (input(azalt) & (i!=0))` komutu ile belirlenen döngü de azalt butonuna basıldığında ve i değişkeni değeri 0 olmadığında aktif olacaktır. Bunun nedeni i değişkeni değeri eğer 0 ise artık azaltma işlemine gerek kalmamasıdır. Bu döngüde de bir önceki döngü de kullanılan aynı komutlar kullanılmıştır. Sadece `i--;` komutu ile bu seferde her butona basılışta i değeri bir azaltılarak 0. potansiyometrenin değeri düşürülmektedir.

Eğer ilk `if (input(pot_sec))` komutunda `pot_sec` anahtarı açık ise bu if döngüsü işlemeyecek `else` komutu altındaki blok işleme girecektir. Bu döngüde de bu sefer aynı mantık ile işlemler 1. potansiyometre için yapılmaktadır.

i ve j değerleri `int i=0,j=0;` komutu ile tam sayı türünde tanımlandığından bu değişken değerleri 256 olunca otomatik olarak 0 değerine geleceklerdir.

22.4. CCS C DERLEYİCİSİ SES DOSYASI

CCS C derleyicisi bünyesinde ses işlemleri için C:\Program Files\PICC\Drivers\ dizini içinde **"TONES.c"** adında bir dosya mevcuttur. Bu dosya kolaylıkla istenen bir ses tonunu üretmemizi sağlayan fonksiyon içermektedir.

generate_tone(frekans, devam süresi); Fonksiyonu = Bu fonksiyon istenen süre boyunca istenen frekansta ses dalgası üretir. Fonksiyonda *"frekans"* kısmına Hz cinsinden istenen frekans değeri girilmelidir. Fonksiyondaki *"devam süresi"* kısmına ise ms (mili saniye) cinsinden istenen süre değeri girilmelidir.

TONES.c dosyası içinde çeşitli oktav değerlerinde müzik tonları tanımlanmıştır. İsterseniz bu tonlar yerine siz kendi müzik tonlarınızı tanımlayabilirsiniz. TONES.c

dosyasını incelediğiniz de bu dosyayı daha iyi tanıyabilirsiniz. Ayrıca CCS C derleyicisi için örnek program dosyalarının yer aldığı C:\Program Files\PICC\Examples\ dizini içinde EX_TONES.c dosyası incelenerek ses programları yapmak için fikir edinilebilir.

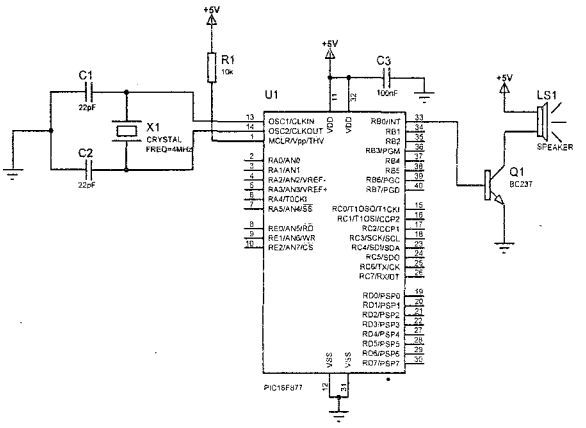
TONES.c dosyasında aşağıda verilen komut ile ses çıkış pin'i RB0 olarak tanımlanmıştır. Bu komuttaki pin ismi değiştirilerek istenen pin ses çıkışı için kullanılabilir.

```
#define TONE_PIN  PIN_B0  // Ses çıkışı RB0 pin'inden olacak
```

22.4.1. Ses Uygulaması

Benzetim: PROTEUS-ISIS ☺ Deneme Kartı: PIC PROG/DEKA 16F87x ☹

Bu uygulamada sadece CCS C derleyicisi içinde bulunan "TONES.c" ses dosyasının kullanıma örnek olması açısından küçük bir program yazılmıştır. Uygulama da RB0 pin'ine bir hoparlör bağlanmıştır. Programda değişik frekansta ve sürede oluşturulan sesler sonsuz döngü içinde sürekli oluşturularak hoparlörden dinlenmektedir. Programda anlamlı bir müzik oluşturulmamıştır. Sadece ses dosyasının kullanıma yönelik olması açısından tüm tonlar kullanılmıştır.



Şekil-22.16. *Ses uygulaması devre şeması.*

Devre programı;

```
/*****************************************************************
      TONES.c dosyasını kullanarak ses elde etme uygulaması.
/*****************************************************************
#include <16f877.h>          // Kullanılacak denetleyicinin başlık
                             // dosyası tanıtılıyor.
// Denetleyici konfigürasyon ayarları
#fuses XT,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOPUT,NOWRT,NODEBUG,NOCPD
#use delay (clock=4000000)   // Gecikme fonksiyonu için kullanılacak
                             // osilatör frekansı belirtiliyor.
```

```
#include <TONES.c>  // TONES.c dosyası programa ekleniyor

// Ses tonlarından oluşan dizi tanımlanıyor
const char muzik[48] = {
C_NOTE[0],C_NOTE[1],C_NOTE[2],C_NOTE[3],Db_NOTE[0],Db_NOTE[1],Db_NOT
E[2],Db_NOTE[3],D_NOTE[0],D_NOTE[1],D_NOTE[2],D_NOTE[3],Eb_NOTE[0],E
b_NOTE[1],Eb_NOTE[2],Eb_NOTE[3],E_NOTE[0],E_NOTE[1],E_NOTE[2],E_NOTE
[3],F_NOTE[0],F_NOTE[1],F_NOTE[2],F_NOTE[3],Gb_NOTE[0],Gb_NOTE[1],Gb
_NOTE[2],Gb_NOTE[3],G_NOTE[0],G_NOTE[1],G_NOTE[2],G_NOTE[3],Ab_NOTE[
0],Ab_NOTE[1],Ab_NOTE[2],Ab_NOTE[3],A_NOTE[0],A_NOTE[1],A_NOTE[2],A_
NOTE[3],Bb_NOTE[0],Bb_NOTE[1],Bb_NOTE[2],Bb_NOTE[3],B_NOTE[0],B_NOTE
[1],B_NOTE[2],B_NOTE[3] };

int i;  // Tam sayı tipinde değişken tanımlanıyor

//******************* ANA PROGRAM FONKSİYONU *******************
void main ()
{
   setup_psp(PSP_DISABLED);          // PSP birimi devre dışı
   setup_spi(SPI_SS_DISABLED);       // SPI birimi devre dışı
   setup_timer_1(T1_DISABLED);       // T1 zamanlayıcısı devre dışı
   setup_timer_2(T2_DISABLED,0,1);   // T2 zamanlayıcısı devre dışı
   setup_adc_ports(NO_ANALOGS);      // ANALOG giriş yok
   setup_adc(ADC_OFF);               // ADC birimi devre dışı
   setup_CCP1(CCP_OFF);              // CCP1 birimi devre dışı
   setup_CCP2(CCP_OFF);              // CCP2 birimi devre dışı

   output_low(pin_b0); // RB0 çıkışı ilk anda lojik-0

   while(1) // Sonsuz döngü
   {
      for(i=0;i<48;i++)
      {
         generate_tone(muzik[i],100); // Sırayla muzık[] dizisindeki
          delay_ms(50);               // tonlar oluşturuluyor
      }
   }
}
/*****************************************************************
```

Programda ilk önce `#include <TONES.c>` komutu ile CCS C derleyicisi müzik tonları dosyası programa eklenmiştir. `const char muzik[48]` komutu ile tanımlanan `muzik[]` dizisi içine TONES.c dosyası içinde tanımlanan tonlar yazılmıştır. İstenirse programcı kendi tonlarını da oluşturabilir. `for(i=0;i<48;i++)` komutu ile oluşturulan for döngüsünde `muzik[]` dizi elemanlarının tümünün sırayla ton üretme fonksiyonuna alınması sağlanmıştır. Fonksiyon döngüsünde `generate_tone(muzik[i],100);` komutu ile `muzik[]` dizisinde yazılan tonlar sırayla 100 ms'lik süre ile oluşturulmaktadır. İstenirse 100 ms'lik süre değiştirilebilir. Yukarıda verilen mantıkla ister hazır tonlar ile istenirse programcı tarafından belli frekans ve sürede oluşturulan tonlar ile istenen ses dosyaları oluşturulabilir.

DİZİN

1

1-WIRE (TEK HAT) İLETİŞİM..... .... 377, 379, 381, 382, 383, 384

2

24C02 EEPROM Entegresi..... 347

7

74138 entegresi..... 152
74164 entegresi..... 150
74C922 Key Enkoder Entegresi ...241,242

8

8255 entegresi 154

A

Adres Operatörü 65
Aritmetiksel Operatörler 54
A/D Kesmesi..... 291
A/D MODÜLÜ 287
ADC modülü..... 288
ADCON0 kaydedicisi..... 288
ADCON1 kaydedicisi..... 289
Asenkron Seri İletişim.....332
ACK.....342, 343
Adım (Step) Motorlar..... 363
ABS( ) 85
ACOS( ) 85
ASIN( ) 85
ATAN( ) 85
ATAN2( ) 85
atof()..... 82
atoi(), atol(), atoi32().....82

B

Brown Out Reset (BOR)..... 27
BackSlash (Ters Eğik Çizgi-\) Sabitleri.. 52
Bit Operatörleri..... 56
BIT_CLEAR() Komutu..... 111
BIT_SET() Komutu..... 111
BIT_TEST() Komutu..... 111

C

CPU..... 11
CCS C.....33, 35, 36
Compile Menüsü 41
C programlama dili..... 48
CCS C Derleyicisinde Operatörler..... 53
CEIL( ) 87
COS( )..... 85
COSH( )..... 85
Continue Komutu..... 128
clear_interrupt();.....182
CAPTURE MODU..... 218
CCP Birimi..... 216
CCPxCON Kaydedicisi..... 217
COMPARE MODU..... 219
CGRAM.....250, 272
CGROM..... 272
CMCON kaydedicisi..... 317
CVRCON kaydedicisi..... 320

D

dsPIC30..... 16
dsPIC33..... 16
Debug Menüsü 44
Document Menüsü..... 45
Değişken Tipleri ve Tanımlamaları..... 50
Dizi Tanımlama..... 60
DELAY_CYCLES() Fonksiyonu..... 129
DELAY_MS() Fonksiyonu 129
DELAY_US() Fonksiyonu..... 129
Do-While Döngüsü.....128
Dot Matris..... 247
Dâhili EEPROM 305
DATA EEPROM 308
DC MOTOR 359
DS1990A i-BUTTON 385
DS1302 .....400, 401
DS1621 .....392, 393, 394, 395, 396
DS1868 .....405, 406

E

Edit Menüsü..... 39
EECON1 kaydedicisi 306
EEPROM Kesmesi (#INT_EEPROM) ... 306
ERASE_PROGRAM_EEPROM() 308
ext_eeprom_ready().....347

F

File Menüsü 38
For Döngüsü 126
FLASH PROGRAM BELLEĞİ.....307
FLASH PROGRAM HAFIZASI..... 308

FGETCH( ) 336
FPUTC( ) 336
FABS( ) 85
FLOOR( ) 87
FMOD( ) 87

G

Göstericiler (pointers) 62
GETENV( ) 81
GOTO_ADDRESS( ) 80
GET_TRIS_X() Komutu 108
goto Komutu 128
GET_TIMERx () Fonksiyonu 192
Grafik LCD 279, 280, 281

H

Harici Osilatör 26
Help Menüsü 47
HD44780 247
HD44780 Tabanlı LCD Komutları 251
Harici EEPROM 346
H-köprü sürücü devresi 359

I

IC-PROG 103
INPUT() Komutu 110
INPUT_STATE() 110
INPUT_X Komutu 110
INTCON Kaydedicisi 176
I^2C 343, 344
I^2C SERİ İLETİŞİM 342
I^2C ve SPI Kesmesi 348
I2C_POLL( ) 346
I2C_READ( ) 345
I2C_START( ) 345
I2C_STOP( ) 345
I2C_WRITE() 345
ISALNUM( ), ISALPHA( ), ISDIGIT( ),
ISLOWER( ), ISSPACE( ), ISUPPER( ),
ISXDIGIT( ), ISCNTRL( ), ISGRAPH( ),
ISPRINT( ), ISPUNCT( ) 83
ISAMOUNG( ) 84

İ

İlişkisel Operatörler 55
İşlemli Atama operatörü 60
if Deyimi 125
if Else Deyimi 125
init_ext_eeprom() 347

K

Kesme Vektörü 20
KESMELER (Interrupts) 33
Konfigürasyon Bitleri 32
Karakter Fonksiyonları 82
Kesme Mantığı 173
Kesme Vektörü 174
KARŞILAŞTIRICI (COMPARATOR)
MODÜLÜ 316
Karşılaştırıcı Kesmesi 318
KARŞILAŞTIRICI VOLTAJ REFERANS
MODÜLÜ 319
KBHIT( ) 337

L

LCD 246, 247, 248, 249
LCD DDRAM Belleği 249
LCD karakter seti 251
lcd_getc(x,y) 258
lcd_gotoxy(x,y) 258
lcd_init () 258
lcd_putc(c) 258
lcd_read_byte() 259
lcd_send_byte() 259
LM35 SICAKLIK SENSÖRÜ 300, 301
L293, L298, L6201, L6202,
L6203,LMD18200 359
L298 DC Motor Sürücü Entegresi 359
L298 Entegresi 360
LABEL_ADDRESS( ) 81
LOG( ) 86
LOG10( ) 86

M

Microcontroller 12
Mikrodenetleyici 12, 13
Mikrodenetleyici kontrol fonksiyonları .. 79
MAKE16() Komutu 112
MAKE32() Komutu 113
Mantıksal Operatörler 56
Matematik Fonksiyonları 85
MAKE8() Komutu 112
Mikroişlemci 11
Matris LED Display 166
MAX232 ENTEGRESİ 334
MAX232 Entegtesi 334

O
Oscillator Start-up Timer (OST) 27
Osilatör Konfigürasyonları 23
Options Menüsü 41
OUTPUT_BIT() Komutu 109
OUTPUT_FLOAT() Komutu 109
OUTPUT_HIGH() Komutu 108
OUTPUT_LOW() Komutu 108
OUTPUT_TOGGLE() Komutu 109
OUTPUT_X() Komutu 109
OPTION Kaydedicisi 177
OPTION Kaydedicisi 188

Ö
Ön İşlemci Fonksiyonları 72
Özel Karakterler 60

P
PIC Mikrodenetleyici Çeşitleri 15
PIC16F877 15
PIC24F 16
PIC24H 16
PORT A 29
PORT B 30
PORT C, PORT D ve PORT E 31
Power On Reset (POR) 27
Program Belleği 19
Program Sayacı 19
PCB 33
PCH 33
PCM 33
Project Menüsü 39
PORT_A_PULLUPS(),
PORT_B_PULLUPS() Komutları 110
PIE1 Kaydedicisi 178
PIE2 kaydedicisi 179
PIR1 kaydedicisi 178
PIR2 kaydedicisi 180
PWM MODU 219
Paralel Slave Port (PSP) Modülü 324
Paralel Slave Port Kesmesi 327
PSP Modülü 324,326,327
PSP_INPUT_FULL() 328
PSP_OUTPUT_FULL() 328
PSP_OVERFLOW() 328
Paralel İletişim 331
PRINTF( ) 337
PUTC( ), PUTCHAR( ) 336
PUTS( ) 337
POW( ) 86

R
Reset vektörü 20
ROM Belleğe Bilgi Yazma 52
Reset Vektörü 174
RESTART_WDT() Fonksiyonu 213
READ_ADC() FONKSİYONU 294
READ_EEPROM() Fonksiyonu 307
READ_PROGRAM_EEPROM() 307
RS232 GİRİŞ/ÇIKIŞ FONKSİYONLARI 336
RS232 SERİ İLETİŞİM 331
RS232 SERİ İLETİŞİM KESMELERİ 335
read_ext_eeprom() 347
R/C SERVO MOTOR 371
READ_BANK( ) 80

S
Sıfırlama (RESET) Devresi 26
Sıfırlaması 28
Search Menüsü 40
Sabit Tanımlama 52
switch-case Şartlı Dallanma Deyimi ... 126
Sayıcı 187
SET_TIMER0,SET_RTCC() Fonksiyonu 191
SET_TIMER1() Fonksiyonu 200
SET_TIMER2() Fonksiyonu 208
SETUP_TIMER_0 () Fonksiyonu 190
SETUP_TIMER_2() Fonksiyonu 207
SETUP_TIMER1() Fonksiyonu 199
SETUP_WDT() Fonksiyonu 21
SET_PWMx_DUTY() Fonksiyonu 222
SETUP_CCPx() Fonksiyonu 221
Seri LCD 255, 256, 257
SET_ADC_CHANNEL() FONKSİYON ... 293
SETUP_ADC() FONKSİYONU 291
SETUP_ADC_PORTS() 292
SET_TRIS_X() Komutu 107
SWAP() Komutu 112
SETUP_COMPARATOR()
FONKSİYONU 319
SETUP_VREF() FONKSİYONU 321
SETUP_PSP() FONKSİYONU 327
Senkron Seri İletişim 332
Seri İletişim 332
SET_UART_SPEED() FONKSİYONU 336
SCK 356
SCL 342, 343
SDA 342, 343

SDI 356
SDO 356
SETUP_SPI( ) 356
SPI SERİ İLETİŞİM..... 356
SPI_DATA_IS_IN( )..... 357
SPI_READ( )..... 357
SPI_WRITE( )..... 357
SS 356
Step (Adım) Motorlar..... 363
SES İŞLEMLERİ..... 410
SIN( ) 85
SINH( )..... 85
SQRT( ) 86
STRCAT( ), STRCMP( ), STRNCMP( ),
STRLEN( ), STRLWR( ) 84
STRCPY( ) 84

T

Tools Menüsü 43
T1CON kaydedicisi 198
T2CON kaydedicisi 206
Timer0 birimi..... 188
Timer1 birimi..... 197
Timer2 birimi..... 205
Timer0..... 189
Timer0 Kesmesi 191
Timer1 Kesmesi 199
Timer2 Kesmesi 207
Tuş Takımı 235,236
TRISE Kaydedicisi.....325
TEK HAT (1-WIRE) İLETİŞİM.....377
TAN( ) 85
TANH( ) 85
TOLOWER( ), TOUPPER( ) 84

U

User ToolBar 46
Unary Operatörler..... 59
ULN2003 Entegresi 365

V

Veri belleği..... 20
View Menüsü..... 42

W

Watch Dog Timer (WDT).....28, 210
While Döngüsü..... 127
WRITE_EEPROM() Fonksiyonu..... 307
WRITE_PROGRAM_EEPROM().....308
write_ext_eeprom().....347
WRITE_BANK( ) 80

Y

Yapılar (structures) 65
Yer Belirleyicileri ve Tanımlamaları 51
Yığın (stack)..... 175

Z

Zamanlayıcı..... 187
Zamanlayıcı/Sayıcı 187

#

#disable_interrupt(); 181
#define use_portb_lcd TRUE 259
#enable_interrupt(); 181
#priority 181
#DEVICE ADC=X KOMUTU.....294
#USE RS232() FONKSİYONU.....335
#USE I^2C () Fonksiyonu 345
#ASM 72
#BIT 78
#BYTE..... 77
#DEFINE 73
#DEVICE 76
#FUSES..... 73
#IF, #ELSE, #ELIF, #ENDIF 75
#IFDEF, #IFNDEF, #ELSE,
#ELIF, #ENDIF 75
#INCLUDE..... 74
#INT_XXXX 74
#LOCATE..... 77
#ORG..... 76
#PRIORITY..... 74
#RESERVE..... 77
#ROM 76
#USE DELAY..... 76
#USE FAST_IO() 78
#USE STANDART_IO()..... 78
#ZERO_RAM 78

KAYNAKÇA

1- "THE I2C-BUS SPECIFICATION VERSION 2.1", PHILIPS Semiconductors
2- 1-Wire Communication Through Software, Application Note 126, Maxim Semiconductor
3- 1-Wire Communication with a Microchip PICmicro Microcontroller, Application Note 2420, Maxim-Dallas Semiconductor
4- 1-Wire Search Algorithm, Application Note 187, Dallas Semiconductor
5- 1-Wire Software Resource Guide Device Description, Application Note 155, Maxim-Dallas Semiconductor
6- 24C02C Serial EEPROM Datasheet, MICROCHIP
7- 2N7000/7002/VQ1000/BS170 N Channel MOSFET DataSheet, VISHAY
8- Add Control, Memory, Security and Mixed-Signal Functions with a Single Contact, Application Note 3938, Maxim Semiconductor
9- Altınbaşak, O., "Mikrodenetleyiciler ve PIC Programlama", Altaş Yayıncılık ve Elektronik, 2001
10- Altınbaşak, O., "Pic Basic Pro İle PIC Programlama", Altaş Yayıncılık ve Elektronik, 2006
11- Arıkan, Ş., "C Programlama Dili", Seçkin Yayıncılık, 2003
12- Aslan, K., "A'dan Z'ye C Klavuzu", Pusula Yayıncılık, 2002
13- BYW98 High Efficiency Fast Recovery Rectifier Diode DataSheet, STMicroelectronics
14- C Compiler Reference Manual Version 4, Custom Computer Services Inc., January 2007
15- Candan, N., "Elektronik-2", Altaş Yayıncılık ve Elektronik, 2003
16- CCS C Compiler Version 4Help File, Custom Computer Services Inc.
17- ÇİÇEK, S., "Renge Göre (Kırmızı, Yeşil, Mavi) Malzeme Taşıyan Robot Kolu Tasarımı ve Uygulaması", Gazi Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans Tez, Ankara, 2006
18- ÇİÇEK, S., C Programlama Dili Ders Notları
19- DM74164 8 Bit Serial In/Parallel Out Shift Register Datasheet, FAIRCHILD SEMICONDUCTOR
20- DM7447A BCD to /-Segment Decoders/Drivers Datasheet, FAIRCHILD SEMICONDUCTOR
21- DS1302 Trickle-Charge Timekeeping Chip Datasheet, Maxim-Dallas Semiconductor, 2005
22- DS1621 Digital Thermometer and Thermostat DataSheet, Maxim-Dallas Semiconductor
23- DS1868 Dual Digital Potentiometer Chip Data Sheet, Maxim-Dallas Semiconductor
24- DS1990A Serial Number iButton DataSheet, Maxim-Dallas Semiconductor
25- Gardner, N., Editör: Elk.Müh.Dinçer, G., Çeviren: Yalçın, C., "Yeni Başlayanlar İçin PIC Programlama El Kitabı", Bileşim ve ERA Ltd., 1998
26- GDM0802A Specification of LCD Module, XIAMEN OCULAR
27- HD44780U(LCD II) Dot Matrix Liquid Crystal Display Controller/Driver, HITACHI

28- HDM64GS12 128x64 Graphics LCD Display Module Product Specification, HANTRONIX
29- ICL7667 MOSFET Driver DataSheet, MAXIM
30- Iovine, J., "PIC Microcontroller Project Book", McGraw Hill, 2000
31- İbrahim., D., "PIC C ile Sıcaklık Projeleri", Bileşim Yayınevi, 2003
32- İbrahim., D., "PIC C Motor Kontrol Projeleri", Bileşim Yayınevi, 2000
33- Karakaş, H., "İleri PIC16F84 Uygulamaları-1", Altaş Yayıncılık ve Elektronik, 2002
34- KS0108B 64CH Segment Driver For Dot Matrix LCD, SAMSUNG ELECTRONICS
35- L298 Dual Full-Bridge Driver DataSheet, ST Microelectronics
36- LM35 Precision Centigrade Temperature Sensors, National Semiconductor
37- LTP-2157AY Matrix, LITE-ON ELECTRONICS INC.
38- MAX620/MAX621 MOSFET Drivers DataSheet, MAXIM
39- MAX8552 MOSFET Driver DataSheet, MAXIM
40- Melabs Serial LCD DataSheet
41- Microchip AN893 Low-Cost Bidirectional Brushed DC Motor Control Using the PIC16F684 Application Note
42- Microchip AN899 Brushless DC Motor Control Using PIC18FXX31 MCUs Application Note
43- Microchip AN905 Brushed DC Motor Fundementals Application Note
44- MM74C922 16 Key Encoder Data Sheet, National Semiconductor
45- MM74HC164 8 Bit Serial In/Parallel Out Shift Register Datasheet, FAIRCHILD SEMICONDUCTOR
46- NetMedia 2x16 LCD Serial LCD Display Module DataSheet
47- Niku, S. B., "Introduction to Robotics : Analysis, System, Applications, 1st ed.", Prentice Hall, New Jersey
48- Overview of 1-Wire Technology and Its Use, Application Note 1796, Maxim Semiconductor
49- Özcerit, A.T., Çakıroğlu, M., Bayılmış, C., "8051 Mikrodenetleyici Uygulamaları", Papatya Yayıncılık, 2005
50- PIC16F87A datasheet, MICROCHIP
51- PIC16F87X datasheet, MICROCHIP
52- PICmicro Mid-Range MCU Family Reference Manual
53- SN54164, SN54LS164, SN74164, SN74LS164 8 Bit Parallel Out Serial In Shift Register Datasheet, TEXAS INSTRUMENTS
54- SN7448 BCD to /-Segment Decoders/Drivers Datasheet, TEXAS INSTRUMENTS
55- Şahin, H., "Bilgisayar Destekli Tasarım PROTEUS", Altaş Yayıncılık ve Elektronik, 2005
56- Şahin, H., Dayanık, A., Altınbaşak, C., "PIC Programlama Teknikleri ve PIC16F877A", Altaş Yayıncılık ve Elektronik, 2006
57- Tan, H., Zhang, J., Luo, W., "Design of DSP Based Multiple Stepping-Motors Micro Stepping Driving Controller", Proceedings of the 2005 IEEE International Conference on Mechatronics, 2005
58- Taşbaşı, G.M., "İleri C Programlama", Altaş Yayıncılık ve Elektronik, 2005

59- Wale, J.D., Pollock, C., "Hybrid Stepping Motors and Drivers", Power Engineering Journal, 2001
60- Zengin, F., Şanlı, M., Urhan, O., Güllü, M.K., "C Dili ile PIC Uygulamaları", Birsen Yayınevi, 2006
61- What Is An iButton?, Application Note 3808, Maxim-Dallas Semiconductor

İnternet Kaynakları

1- 24CXX SERİ EEPROMLAR, http://www.antrak.org.tr/gazete/122003/ziya_erdemir.htm, Ziya Erdemir
2- http://elkom-serwis.com.pl
3- http://gleasonresearch.com/prod.php?sku=CX-MOTOR
4- http://homepage1.nifty.com
5- http://kudelsko.free.fr/prog_pic_rs232/presentation2.htm
6- http://lcd.sourceforge.net
7- http://www.153korea.co.kr/pro/pro_25_01.asp
8- http://www.comfile.co.kr/product/image/prod_im10_etcs11.gif
9- http://www.csharpnedir.com/makalegoster.asp?MId=598
10- http://www.elektroboard.com/v2/modules/smartsection/print.php?itemid=4
11- http://www.leds-buy.nl
12- http://www.linengineering.com/site/BLDC/BL24A.htm
13- http://www.maxim-ic.com/products/1-wire/
14- http://www.orientalmotor.com/images/in_motion/step_motor_cutaway.jpg
15- http://www.pcgadgets.com/DC%20motor.jpg
16- http://www.piclist.com/techref/io/lcd/panel1.htm
17- http://www.us.oup.com/us/companion.websites/0195142519/studentresources/diode/?view=usa, Oxford University Press USA
18- http://www.uydutvhaber.net
19- www.acroname.com
20- www.active-robots.com/.../matrix-orbital.shtml
21- www.imagesco.com/articles/lcd/03.html
22- www.nokytech.net/forum/showthread.php?t=39771
23- www.smarthome.com/78550.html
24- www.snailinstruments.com