PIC MİKRODENETLEYİCİLERİ İÇİN

mikroC

DERLEYİCİSİ

KULLANIM VE UYGULAMA KİTABI

Çeviri Ekibi:

Hakan ATBAŞ
B. Sc. EEE, ODTÜ/1994
Programlama Dilleri Uzmanı

Dr. F. Zeynep KÖKSAL
Ph.D. EEE, ODTÜ/1990
Gömülü Sistemler Uzmanı

Kamuran SAMANCI
B. Sc. EEE, Ank.Üni./2006

BETİ BİLİŞİM TEKNOLOJİLERİ LTD. ŞTİ.
MAYIS 2009

1. Baskı

Editör:

Hakan ATBAŞ
B. Sc. EEE, ODTÜ/1994
Programlama Dilleri Uzmanı

İsteme Adresi:

Beti Bilişim Teknolojileri İmalat
Sanayi İç ve Dış Tic. Ltd. Şti.
Şerefli Sokak No:40/5
Mebusevleri/Ankara
Tel: 0-312-222 18 00
Faks: 0-312-222 18 08

Baskı :
Başak Matbaacılık ve Tan. Hiz. Ltd. Şti.
Anadolu Bulvarı No: 5/15
Yenimahalle - Gimat / ANKARA
Tel : 0 312 397 16 17

ISBN = 978-9944-5821-1-7

ÖNSÖZ

PIC mikrodenetleyicilerinin gömülü (embedded) elektronik alanındaki uygulamalarında, mikroBasic derleyicisini kullanarak kolay kod geliştirme ile başlayan teknolojik yolculuğumuza C derleyicisi ile devam ediyoruz.

Gömülü sistemlerin programlanmasında uzun yıllar boyunca makina dili kullanılmıştır. Ancak makina dili öğrenmesi zor bir dildir. Ayrıca program büyüklüğü arttıkça makina dili ile yazılan programların kontrolü de oldukça güçleşir.

C programlama dili profesyonel dünyada kendini ispatlamış, öğrenmesi makina diline göre kolay, evrensel ve verimli bir programlama dilidir.

mikroC derleyicisi, C dilini ve PIC mikrodenetleyicilerini gayet uyumlu bir şekilde bir araya getirip size sunarak büyük bir iş başarıyor. Tüm PIC uygulamalarınızı mikroC derleyicisini kullanarak hızlıca ve kolayca gerçekleştirebilirsiniz. PIC12, PIC16 ve PIC18 aileleri için gereken kodları mikroC'nin gelişmiş Windows tabanlı IDE'si ile "işaretle ve tıkla" kadar kolayca hazırlayabilirsiniz.

Bu kitabın aşağıdaki hedef kitle için gayet yararlı olacağına inanıyoruz:

C programlama dilinde tecrübesi olup PIC mikrodenetleyicileri üzerinde makina dilinin zorlukları ile uğraşmadan, kolayca uygulama geliştirmek isteyenler,

Hem geçerli bir bilgisayar dilini öğrenmek hem de gömülü sistemler üzerinde özellikle de PIC programlama konusunda tecrübe sahibi olmak isteyenler,

İşi gereği gömülü sistemler üzerinde çalışan mühendisler, tekniker ve teknisyenler,

Duyarlı akademisyenler, elektronik, elektromekanik ve bilgisayar öğretmenleri,

Gömülü sistemler üzerinde daha önceden makina dili, BASIC, Pascal gibi dillerde uygulama geliştirmiş ve kendisini yetiştirmek isteyen uzmanlar,

Elektronik, bilgisayar veya elektromekanik eğitimi gören her seviye öğrenciler,

Amatör elektronik meraklıları.

mikroC derleyicisi'nin 2K demo versiyonunu içeren CD ile birlikte:

Birçok uygulamaya yönelik hazır alt-yordamlar, özellikle I2C, SPI, CAN gibi güncel teknolojik cihazlarda yoğun olarak kullanılan arabirimler, karakter ve grafik LCD'ler, dokunmatik paneller,

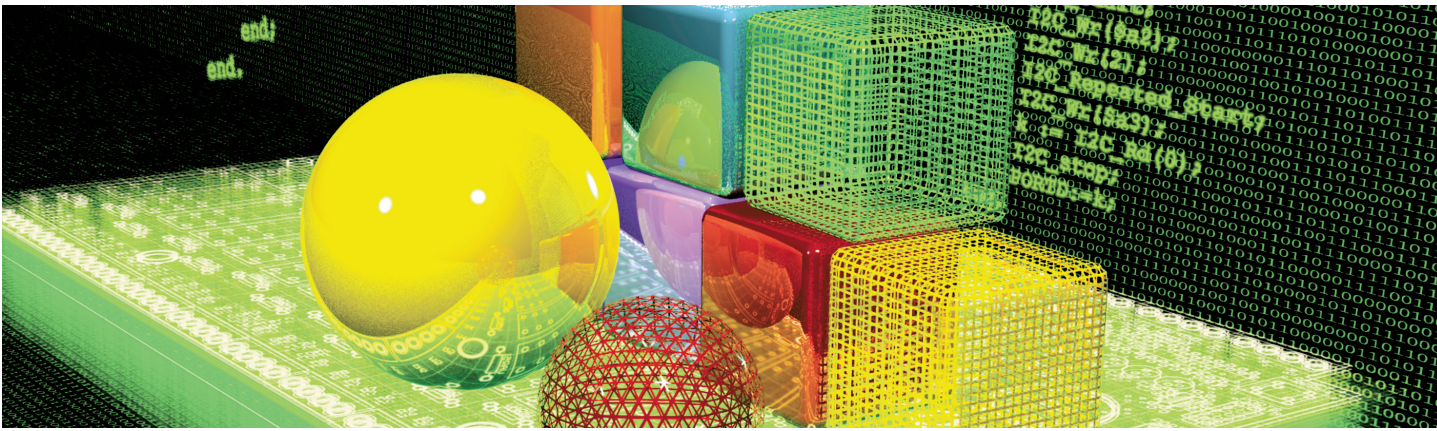
Güncel teknoloji uygulamalarını kolaylaştıran çeşitli kod örnekleri;

Geniş içerikli help menüsü ve …

Yaklaşık 424 sayfalık titizlikle hazırlanmış Türkçe Kullanım Kitabı'nı Beti Bilişim gururla beğeninize sunar.

/ Mayıs 2009

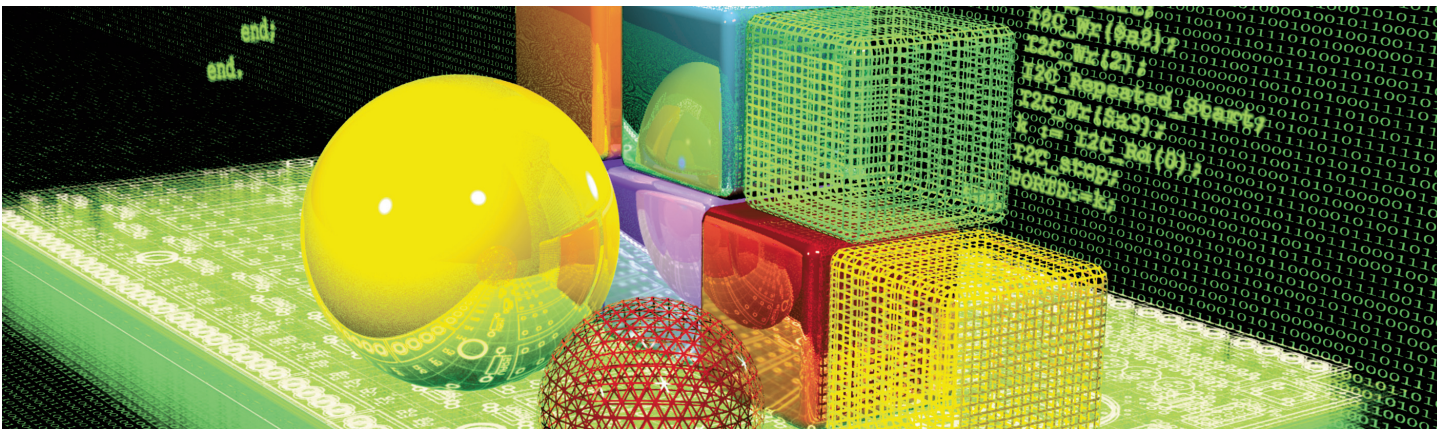
mikroC



İÇERİK

İÇİNDEKİLER

BÖLÜM 1

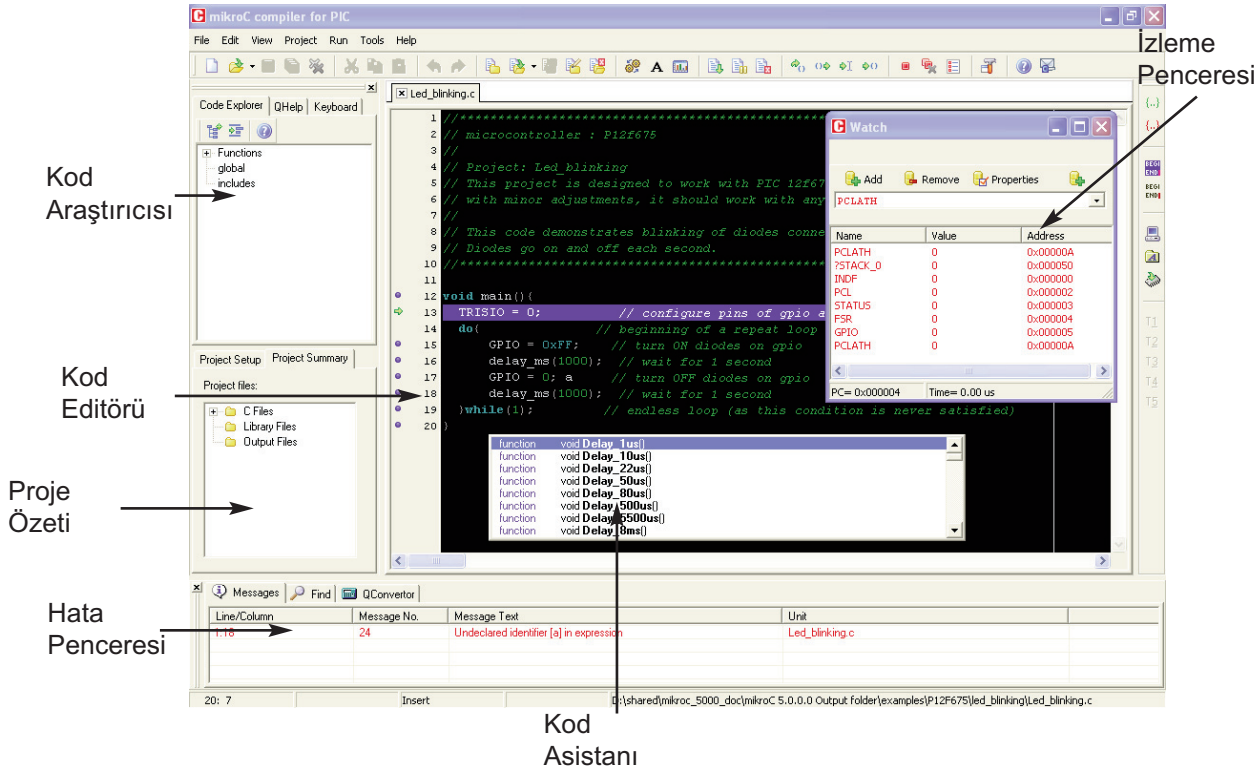


mikroC IDE

GİRİŞ

mikroC PIC microdenetleyicileri için güçlü ve zengin özellikli bir geliştirme aracıdır. Program geliştiricilerin, performans ve kontrolden taviz vermeden, gömülü sistemler için uygulama geliştirmeleri için en kolay çözümü sağlamak amacı ile geliştirilmiştir.

PIC ve C birbirleri ile çok uyumludurlar: PIC, dünyada en popüler 8-bitli yongadır ve çok geniş uygulama alanlarında kullanılmaktadır. Öte taraftan C, verimli ve etkili bir programlama dili olduğu için gömülü sistemler için en doğal tercihtir. mikroC; çok gelişmiş özelliklere sahip tümleşik geliştirme ortamı (IDE), ANSI uyumlu derleyici, geniş donanım kütüphanesi seti, kapsamlı doküman ve bol miktarda çalışmaya hazır örnekleri ile size iş arkadaşı olmaya hazırdır.



mikroC karmaşık uygulamaları çabucak geliştirebilmenizi sağlar:

- Gelişmiş Kod Editörünü kullanarak C kaynak kodlarını yazın.

- Dahili mikroC kütüphanelerini kullanarak uygulamalarınızı çok daha hızlı bir şekilde geliştirin: Örneğin veri toplama, bellek işlemleri, ekranlar, dönüştürmeler, haberleşme ...

- Kod Araştırıcı'sından programınızın yapısını, değişkenlerini ve fonksiyonlarını izleyin. Kod yorumlarınızın da içine eklendiği sembolik makina dili kodları üretin. Tüm standard programlayıcılarda kullanabileceğiniz HEX kodlarını üretin.

- Dahili Hata Denetleyicisi (Debugger) ile program akışını gözleyin ve mantıksal hataları tesbit edin. Kod istatistiği, sembolik makina dili listesi ve fonksiyon dallanma ağacı (calling tree) hakkında detaylı rapor ve grafikler edinin.

- Projelerinizi geliştirebilmeniz, genişletebilmeniz ve çeşitli projelerinizde yapı taşları olarak kullanabilmeniz için bol miktarda örnek hazırladık.

KOD EDİTÖRÜ

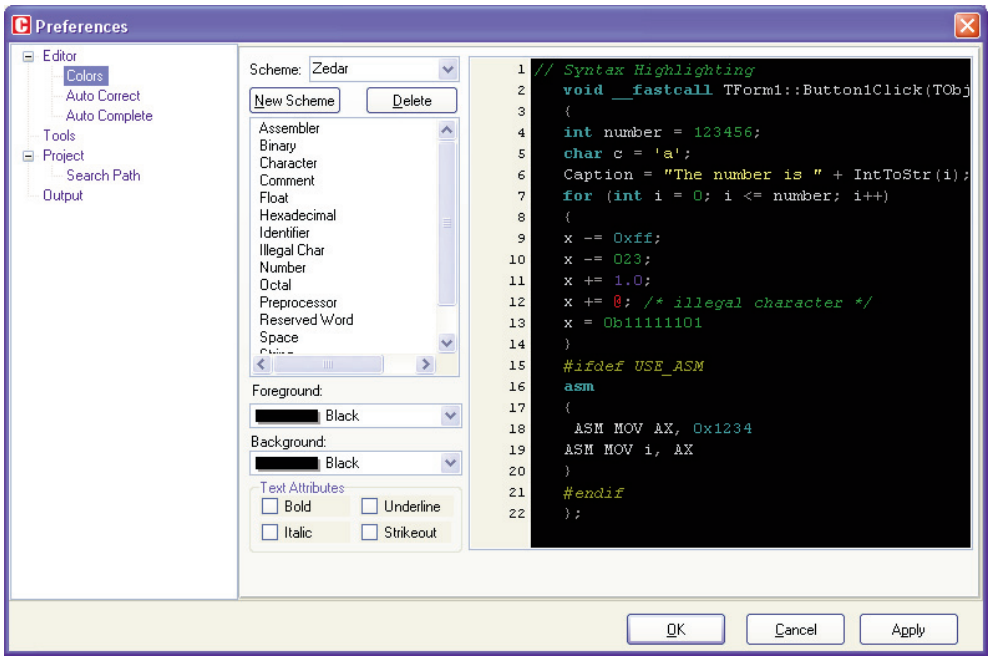
Kod Editörü, profesyonellerin ihtiyaçlarını tatmin edebilecek şekilde geliştirilmiş ileri düzeyde bir metin editörüdür. Kod Editörü standart metin editörleri gibi çalışmaktadır; Windows ortamında yakından bildiğimiz gibi Copy, Paste ve Undo... gibi tanıdık komutları kullanmaktadır.

Geliştirilmiş Kod Editörü aşağıdaki özellikleri içerir:

- Ayarlanabilir sözdizimi işaretleme
- Kod Asistanı
- Parametre Asistanı
- Kod Şablonları (Otomatik Tamamlama)
- Otomatik Düzeltme
- Yer imleci ve Goto Line

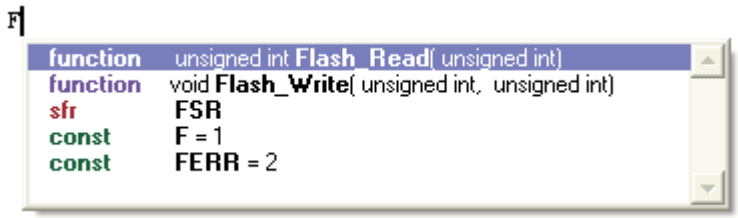
Ayrıca bu tür opsiyonları *Preferences* yani tercihler penceresinden kişiselleştirebilirsiniz. Bu işlem için ya açılır menüden **Tools > Options**'a tıklayın veya sadece *Tools* ikonu'na tıklayın veya da klavyenizdeki F12 tuşuna basın.

Tools İkonu.



Kod Asistanı [CTRL+SPACE]

Bir kelimenin birkaç harfini yazdıktan sonra CTRL+SPACE'e basınız, yazdığınız harfleri içeren tüm geçerli tanıtıcılar (örneğin fonksiyon adları, sabit veya değişken adları) yazdığınız panelde görülecektir (alttaki resme bakınız). Bundan sonra ya seçimi daraltmak için diğer harfleri yazmaya devam ediniz veya klavyeden seçiminizi yapıp Enter'a basınız.



Parametre Asistanı [CTRL+SHIFT+SPACE]

Parametre Asistanı bir parantez açıldığında "(" veya CTRL+SHIFT+SPACE'e basıldığında otomatik olarak başlıyacaktır. Eğer fonksiyon veya yordam ismi parantezden önce gelirse, fonksiyon için geçerli parametreler size kayan bir panelde gösterilecektir. Geçerli parametreyi yazdığınızda bir sonraki beklenen parametre koyulaşacaktır.

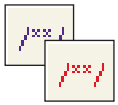
Kod Şablonu [CTR+J]

Kod Şablonlarını, şablonların isimlerini yazarak kodunuza ekleyebilirsiniz, örneğin hazır şablonlardan biri olan while statement (while deyimi) kalıbını kodunuza eklemek için : *whiles* yazın, daha sonra CTRL+J'ye basın, Editor kodu otomatik olarak üretecektir.

Kendi şablonlarınızı şablon listesine ekleyebilirsiniz. Bunun için menüden **Tools>Options**'ı seçin, veya *Settings Toolbar*'dan (Ayarlar Araç Çubuğu'ndan) *Tool*s ikonunu (üzerinde çekiç resmi olan ayarlar butonu) tıklayın ve *Auto Complete Tab*'ı seçin. Buradan uygun anahtar kelimeyi, tanımı ve şablon kodunu girebilirsiniz.

Otomatik Düzeltici (Auto Correct)

Otomatik düzeltici bazı genel yazım hatalarını düzeltir. Tanımlı kelime listesine girmek için açılır menüden, **Tools> Options**'ı seçin veya *Settings Toolbar*'dan *Auto Correct Tab*'ı tıklayın. Kendi tercihlerinizi de listeye ekleyebilirsiniz.



Comment / Uncomment Ikonu.

Yorum/Yorum değil (Comment/Uncomment)

Kod Editörü size bir kod bloğundan bir bölümü "yorum" veya "yorum değil" olarak işaretlemeyi basit bir fare tıklaması ile sunmaktadır. Bu işlem için *Code Toolbar*'da bulunan *comment/uncomment* ikonlarını kullanmalısınız.

Yer İmleri (Bookmarks)

Yer imleri uzun kodlarda gezinmeyi kolaylaştırır:

CTRL+<number> : Yer imine (Bookmark'a) git
CTRL+SHIFT+<number> : Ekranda buraya ilgili yer imini (Bookmark'ı) koy

Satıra Git (Goto Line)

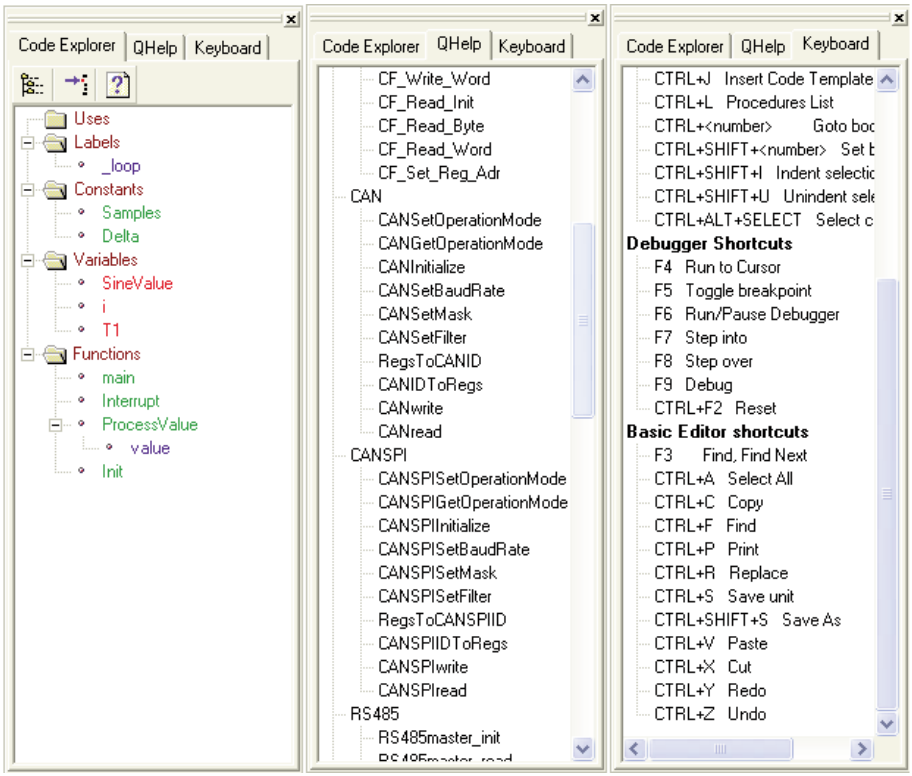
Satıra git (Goto Line) seçeneği satır numarası girerek ilgili satıra gitmeyi sağlar. Böylelikle uzun kodlar içerisinde gezinmeyi kolaylaştırır. Bu işlem için CTRL+G kısa yolunu kullanınız.

KOD ARAŞTIRICISI

Kod Araştırıcısı ana pencerenin sol tarafına yerleştirilmiştir ve kaynak kodundaki tüm tanımlı elemanlar hakkında ayrıntılı bilgi verir. Herhangi bir öğenin ayrıntısına ögenin üzerine sağ tıklayarak geçebilirsiniz veya *Find Declared* ikonunu tıklayarak gerçekleştirebilirsiniz. Kod Araştırıcısı içerisindeki ağaç yapısını (treeview) daraltmak veya genişletmek için *Collapse/Expand* ikonu kullanılabilir.

Collapse/Expand All İkonu.

Kod Araştırıcısı içerisinde ayrıca iki tab penceresi daha vardır. *QHelp Tab*, çabuk ulaşım için tüm yerleşik ve kütüphane fonksiyonlarını listeler. Qhelp içerisindeki bir yordamın çift tıklanması ile, ilgili yardım başlıkları açılır. *Klavye Tab* (Keyboard Tab) mikroC içerisindeki tüm klavye kısayollarını listeler.



HATA AYIKLAYICI (DEBUGGER)

Hata ayıklayıcı (Debugger) mikroC tarafından geliştirilmiş bir tümleşik bileşendir. Microchip Technology'nin PICmicro'larının işlemlerini taklit etmek ve bu yongalar için yazılmış yazılımların hatalarını tespit etmek için geliştirilmiştir.

Debugger program akışını ve komutların yürütülmesini taklit eder, fakat PIC'in işlevlerini tam olarak gerçekleştiremez; örneğin zamanlayıcı ve kesme bayraklarını güncellemez v.b.

Projenizi sorunsuz bir şekilde derledikten sonra, Debugger'i açılır menüden **Run>Debug'ı** seçerek veya *Debug* ikonuna tıklayarak çalıştırabilirsiniz. Debugger'ın etkinleştirilmesi ile daha başka seçenekler de etkinleşir: *Step into, Step Over, Run to Cursor* vs. İşlenecek komut satırı ise koyulaştırılır.


Start Debugger

Debug [F9]
Debugger'ı başlat.


Pause Debugger

Run/Pause Debugger [F6]
Debugger'ı çalıştır veya çalışıyor ise duraklat.


Stop Debugger

Stop Debugger [Ctrl+F2]
Debugger'ı durdur.


Step Into

Step Into [F7]
Geçerli C komutunu (tekli-çoklu-döngü) çalıştır, daha sonra dur. Eğer komut bir yordama dallanma ise, yordama gir ve dallanma komutundan sonraki komutta dur.


Step Over

Step Over [F8]
Geçerli C komutunu (tekli-çoklu-döngü) çalıştır, daha sonra dur. Eğer komut bir yordama dallanma ise, atla ve dallanma komutundan sonraki komutta dur.


Step Out

Step Out [Ctrl+F8]
Geçerli C komutunu (tekli-çoklu-döngü) çalıştır , daha sonra dur. Eğer komut bir yordamın içerisinde ise, komutu çalıştır ve dallanma komutundan sonraki komutta dur.

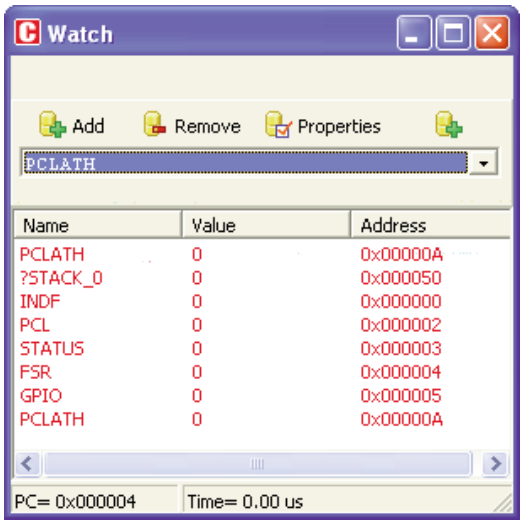

Run to Cursor

Run to cursor [F4]
Geçerli komut ve imleç arasındaki tüm komutları çalıştır.

Toggle Breakpoint.

Toggle Breakpoint [F5]
İmleç konumunda durma-noktası (breakpoint) koyar/kaldırır (toggle). Tüm durma-noktalarını görmek için, açılır menüden **Run>View Breakpoints**' i seçin. Açılan penceredeki değerlerden birine çift tıklandığında durma noktasının yeri görüntülenir.

Gözetleme Penceresi (Watch Window)
Hata Ayıklayıcı Gözetleme Penceresi (Debugger Watch Window) çalıştırdığınız programınızın program parçalarını gözetlemenizi sağlayan bir ana hata ayıklayıcı penceresidir. Gözetleme penceresini görmek için, açılır menüden **View>Debug Windows>Watch Window**'u seçebilirsiniz.



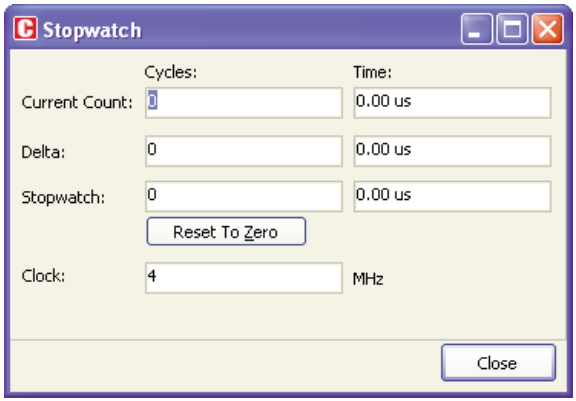
Watch Window PIC'nin değişkenlerini ve kayıtçılarını onların adres ve değerleri ile birlikte gösterir. Simulasyonu çalıştırınca değerler güncellenir. Gözlemek istediğiniz parametreyi açılır menüden ekleyebilir veya çıkartabilirsiniz. En son seçilen parametreler kırmızı ile renklendirilmiştir.

Herhangi bir parametreye çift tıklanınca *Edit Value* penceresi açılacaktır. Bu pencerede seçtiğiniz değişken veya kayıtçının değerini değiştirebilirsiniz. Aynı zamanda bu parametrenin ikilik, onluk, onaltılık, karakter veya ondalık olarak görünümünü seçebilirsiniz.

Kronometre Penceresi (Stopwatch Window)

Hata Denetleyici Kronometre Penceresi (Debugger Stopwatch Window) açılır menüden, **View > Debug Windows > View Clock** seçilerek aktif hale getirilir.

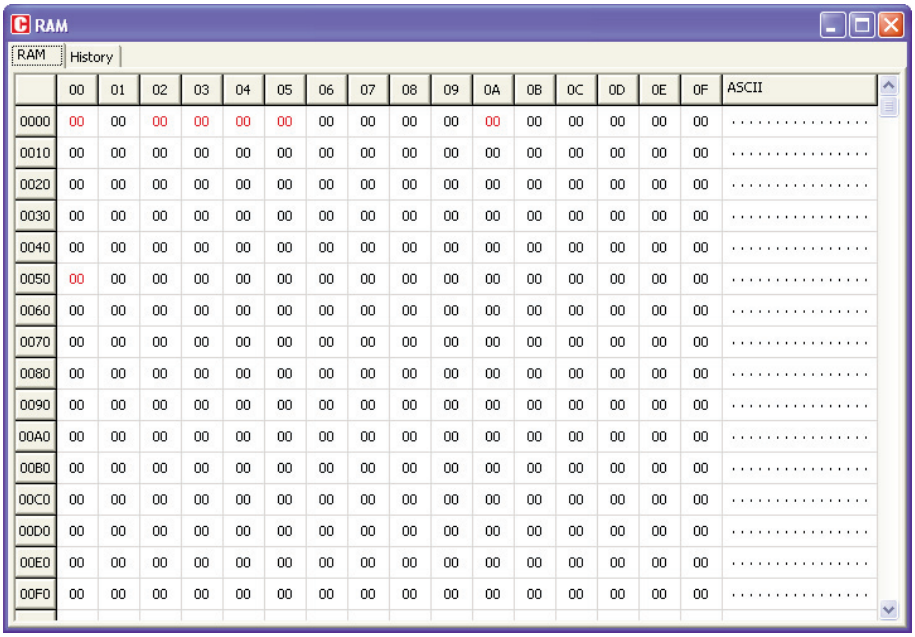
Kronometre penceresi son hata ayıklama işleminden sonraki geçerli çevrim/zaman sayısını gösterir. Kronometre hata ayıklama başladığı andan itibaren çalışma süresini (çevrim sayısı) ölçer ve herhangi bir anda sıfırlanabilir. *Delta* bir önceki komuttan (hata ayıklamanın başlatıldığı) şu an aktif olan komuta (hata ayıklamanın sonlandırıldığı) kadar olan çevrim sayısını gösterir.



Not: Kronometre içerisindeki saati (clock) değiştirebilirsiniz; bu durumda değerler yeni belirtilmiş frekans için yeniden hesaplanacaktır. Kronometre penceresindeki saatin değiştirilmesi esas proje ayarlarını etkilemeyecektir, sadece bir benzeşim sağlayacaktır.

Rastgele Erişimli Bellek (RAM) Görünüm Penceresi (View RAM Window)

Debugger'ın RAM Görünüm Penceresi (View RAM Window) açılır menüden **View > Debug Windows > View RAM** seçilerek aktif hale getirilir.
View RAM penceresi en son kırmızı işaret ile seçilmiş PIC'in RAM bellek haritasını gösterir. Herhangi bir alanın değerini üzerine çift tıklayarak değiştirebilirsiniz.

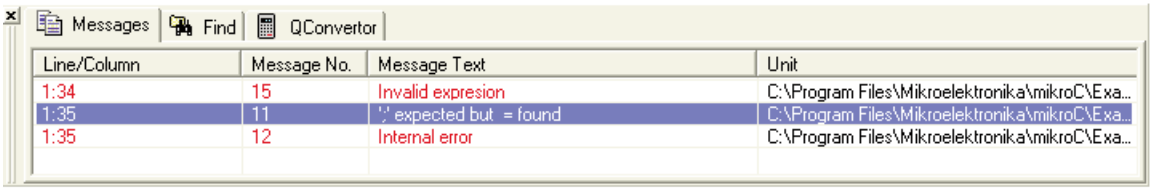


	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

HATA PENCERESİ

Derleme esnasında hata ile karşılaşıldığında, derleyici onları raporlar ve herhangi bir hex dosyası üretmez. Hata penceresi ana pencerenin altında görüntülenir.

Hata Penceresi mesaj sekmesinin altına yerleştirilmiştir ve derleyicinin bulduğu hataların tiplerini ve yerlerini gösterir. Derleyici aynı zamanda uyarıları da rapor eder, ancak uyarılar hex kod üretilmesine engel değildir.



Hata penceresinde mesaj satırına çift tıklandığında hatanın nerede olduğu gösterilecektir.

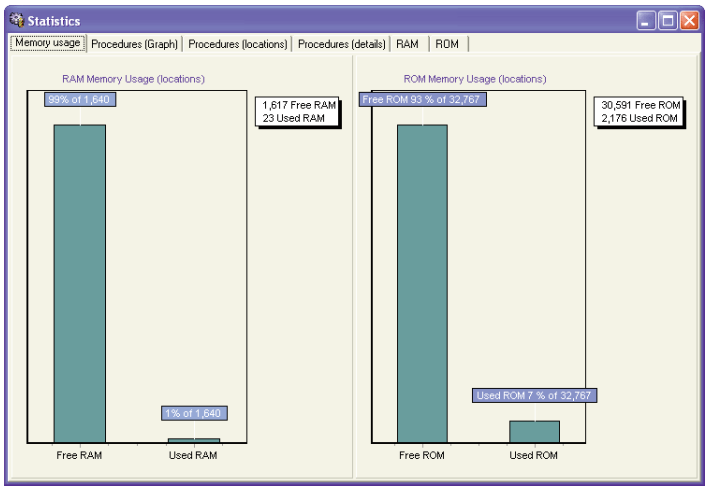
Derleyicinin tanıdığı hatalar hakkında daha fazla bilgi edinmek için “Hata Mesajları” bölümüne bakınız.

İSTATİSTİKLER

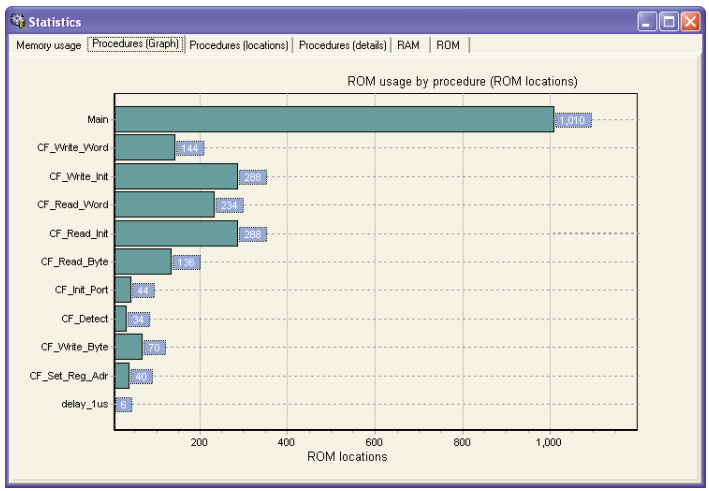
Statistics İkonu.

Hatasız bir derleme işleminden sonra kodunuzun istatistiğini görebilirsiniz.Bu işlem için açılır menüden **Project > View Statistics** 'i seçin veya *Statistics* ikonuna çift tıklayın. İstatistiklerde altı tane sekme penceresi mecuttur:

Bellek Kullanma Penceresi
RAM ve ROM bellek kullanımını histogramlar halinde gösterir.

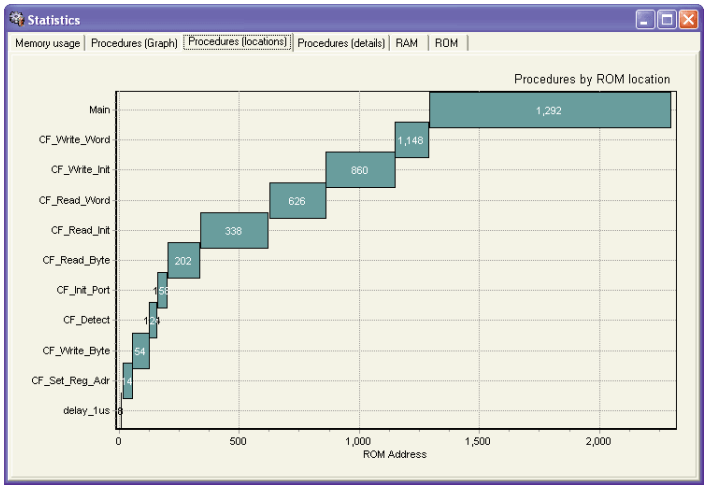


İşlevlerin Grafik Penceresi (Procedures (Graph) Window)
İşlevleri (fonksiyonları) bellek kullanımına göre histogram şeklinde gösterir.



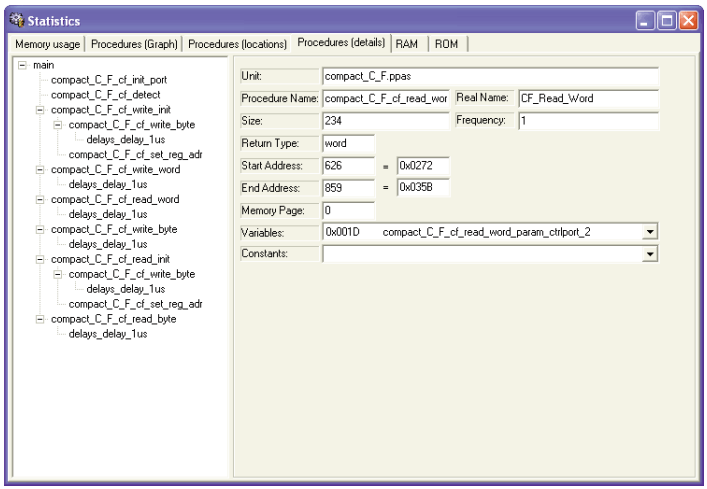
İşlevlerin Yerleşimi Penceresi (Procedures (Locations) Window)

İşlevlerin (fonksiyonların) mikrodenetleyici belleğinde nasıl dağıldıklarını gösterir.



İşlevlerin Detayları Penceresi

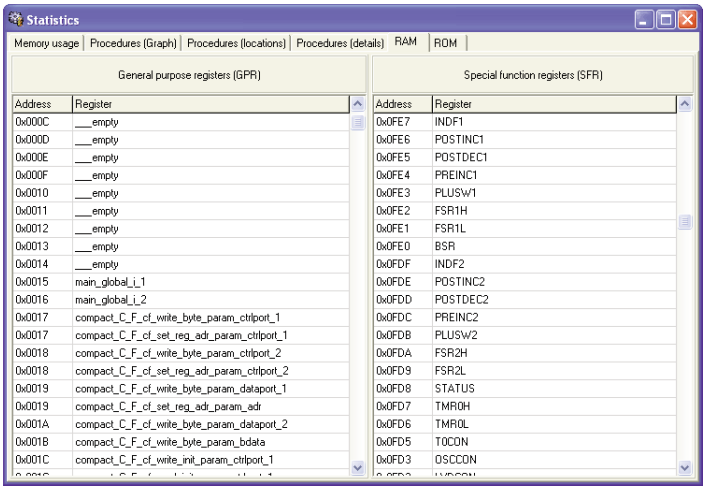
Her işlevin (fonksiyonun) detayları ile birlikte komple dallanma ağacını gösterir:



Boyut , başlama ve bitiş adresleri, dallanma frekansları, dönüş tipi v.b.

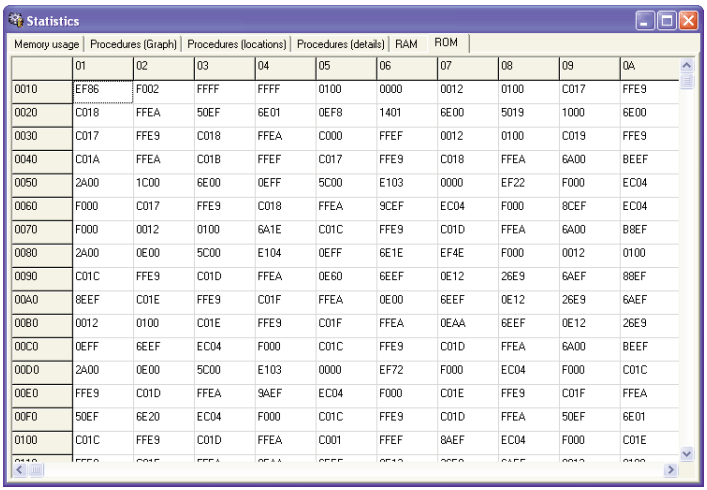
Rastgele Erişimli Bellek (RAM) Penceresi (RAM Window)

Tüm GPR ve SFR kayıtçılarını ve onların adreslerini özetler. Ayrıca değişkenlerin ve bu değişkenlerin adreslerinin sembolik isimlerini gösterir.



Salt Okunur Bellek (ROM) Penceresi (ROM Window)

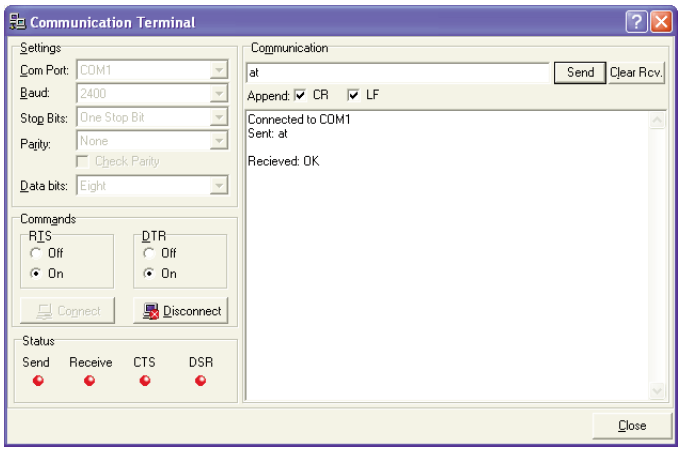
Op-code'ları ve adreslerini okunabilir bir hex kod halinde listeler .



TÜMLEŞİK ARAÇLAR (INTEGRATED TOOLS)

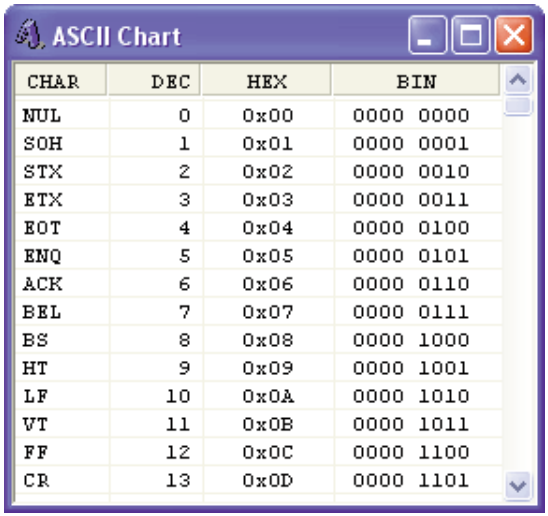
USART Terminali

mikroC RS232 seri haberleşmesi için USART (Universal Synchronous Asynchronous Receiver Transmitter) haberleşme terminalini içerir. Terminal'e, açılır menüden **Tools > Terminal'i** seçerek veya *Terminal* ikonuna tıklayarak ulaşılabilir.



ASCII Tablosu (ASCII Chart)

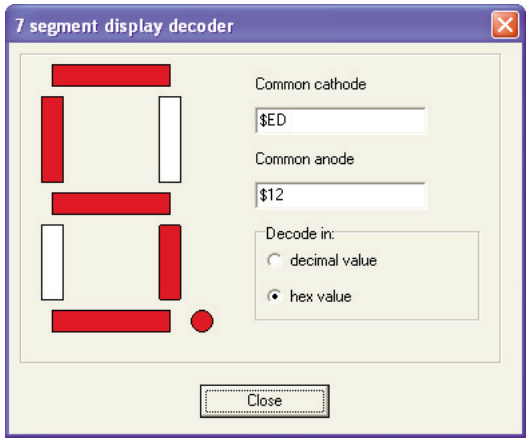
ASCII tablosu hazır ve pratik bir araç olup, özellikle LCD gösterge uygulamalarında çok kullanışlıdır. Tabloya açılır menüden **Tools > ASCII chart** seçilerek ulaşılabilir.



CHAR	DEC	HEX	BIN
NUL	0	0x00	0000 0000
SOH	1	0x01	0000 0001
STX	2	0x02	0000 0010
ETX	3	0x03	0000 0011
EOT	4	0x04	0000 0100
ENQ	5	0x05	0000 0101
ACK	6	0x06	0000 0110
BEL	7	0x07	0000 0111
BS	8	0x08	0000 1000
HT	9	0x09	0000 1001
LF	10	0x0A	0000 1010
VT	11	0x0B	0000 1011
FF	12	0x0C	0000 1100
CR	13	0x0D	0000 1101

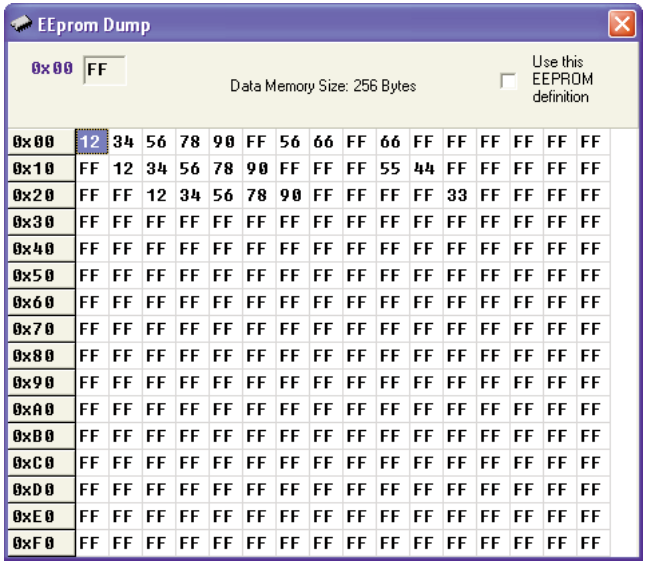
7- Parçalı Gösterge Kod Çözücüsü (7-Segment Display Decoder)

7 Segment Display Decoder ondalık/hex değerleri uygun kombinasyonlar halinde gösteren görsel bir paneldir. 7-parçalı gösterge sembolündeki parçalara tıklayın ve seçiminize karşılık gelen değerleri editör kutusunda görün. 7-parçalı göstergeye açılır menüden; **Tools > 7 Segment Converter'ı** seçerek ulaşabilirsiniz.



EEPROM Editörü (EEPROM Editor)

EEPROM Editörü, kullanıcının PIC microdenetleyicisinin EEPROM'unu kolayca yönetmesine yarar.



0x00	12	34	56	78	90	FF	56	66	FF	66	FF	FF	FF	FF	FF	FF
0x10	FF	12	34	56	78	90	FF	FF	FF	55	44	FF	FF	FF	FF	FF
0x20	FF	FF	12	34	56	78	90	FF	FF	FF	FF	33	FF	FF	FF	FF
0x30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xA0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xB0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xC0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xD0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xE0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0xF0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

mikroÖnyükleyicisi (mikroBootloader)

mikroBootloader sadece *flash write*'ı destekleyen PIC mikrodenetleyicileri ile birlikte kullanılabilir. mikroBootloader kullanırken şu adımları izleyiniz:

1. PIC'e geleneksel programlama tekniği ile uygun hex bootloader dosyasını yükleyin (Örneğin PIC16F877A için pic16f877a.hex kullanın).
2. Açılır menüden **Tools > Bootloader'ı** tıklayarak mikroBootloader'ı başlatın.
3. Setup Port'a tıklayın ve kullanılacak COM (seri) portunu seçin. BAUD'un 9600 Kbps'a ayarlı olduğuna emin olun.
4. Open file'a tıklayın ve bootloader aracılığı ile yüklenecek HEX dosyasını seçin.
5. PIC'teki önyükleme kodu bilgisayara bağlanmak için yalnızca 4-5 saniye süre tanıdığından bu süre içinde PIC'i resetleyin ve Connect butonuna basın.
6. Geçmiş (History) penceresinin en son satırı şu anda "Connected" göstermelidir.
7. Yüklemeyi başlatmak için, *Start Bootloade*r'a tıklanması gerekir.
8. Programınız PIC'in flash belleğine yazılacaktır. Yükleme esnasında hata oluşursa Bootloader bunu raporlayacaktır.
9. PIC'i resetleyin ve programınızı çalıştırmaya başlayın.

PIC'teki önyükleme kodu bilgisayara bağlanmak için 5 saniye verir, bağlanılmazsa, PIC'teki mevcut kullanıcı kodunu çalıştırmaya başlar. Eğer bir kullanıcı kodu yüklenecekse, önyükleyici kodu alır ve bu veriyi program belleğine yazar.

Bootloader'ın daha genel özellikleri aşağıda listelenmiştir:
- Reset bölgesindeki kodun güncellenmesi .
- Belleğin herhangi küçük bir bölgesindeki kodun güncellenmesi.
- Kullanıcının yeni bir kod yüklemek isteyip istemediğinin kontrol edilmesi.
- Yeni bir kullanıcı kodu yüklenmemişse mevcut yüklü kullanıcı kodunun çalıştırılmaya başlanması.
- Yeni kod yüklecek ise bir haberleşme kanalından kullanıcı kodunun alınması.
- Yeni kullanıcı kodunun belleğe kaydedilmesi.

Kullanıcı ve Önyükleme Kodunun Tümleştirilmesi
Önyükleme kodu genelde reset bölgesini ve bazı ek program bellek alanını kullanır. Kesmelere ihtiyacı olmayan basit bir kod parçasıdır. Böylelikle, kullanıcı kodu 0x0004'teki normal kesme vektörünü kullanabilir. Önyükleme kodu kesme vektörü kullanmaktan kaçınmalıdır. Bu nedenle 0x000 ile 0x0003 adresleri arasında bir program dallanması yapması gerekir. Önyükleme kodu geleneksel programlama teknikleri ile bellek bölgesine konfigürasyon bitleri ile aynı zamanda programlanmalıdır. Önyükleme kodu konfügrasyon bitlerine ulaşamaz, zira bu bitler program hafıza alanına eşlenmemiştir.

KLAVYE KISA YOLLARI

Aşağıda mikroC IDE içerisindeki geçerli olan tüm klavye kısa-yolları verilmiştir. *Code Explorer*'dan klavye'ye sekme yaparak klavye kısa-yollarına ulaşabilirsiniz.

IDE Kısayolları

F1	Yardım
CTRL+N	Yeni dosya
CTRL+O	Dosya veya proje aç
CTRL+F9	Derle
F11	Mikrodenetleyiciyi programla
CTRL+F11	Derle ve mikrodenetleyiciyi programla
F12	Tercihler (Preferences) penceresini göster

Temel Editör Kısayolları

CTRL+A	Tümünü seç
CTRL+C	Kopyala
CTRL+F	Bul
F3	Sonrakini bul
CTRL+P	Yazdır
CTRL+H	Bul ve değiştir
CTRL+S	Kaydet
CTRL+V	Yapıştır
CTRL+X	Kes
CTRL+Y	Tekrar yap (Redo)
CTRL+Z	Geri al (Undo)

Gelişmiş Editör Kısayolları

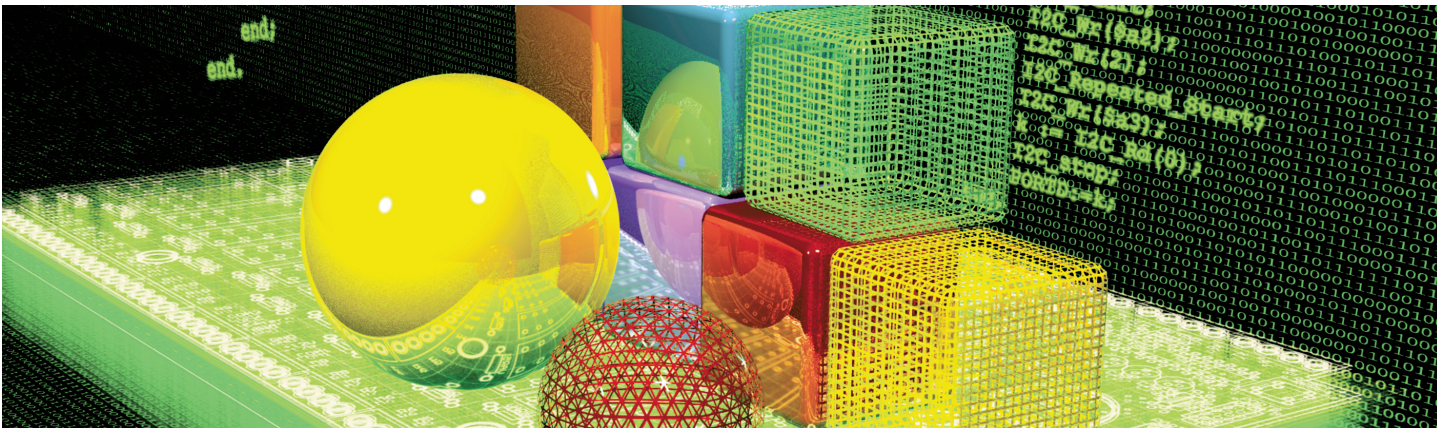
CTRL+SPACE	Kod yardımcısı
CTRL+SHIFT+SPACE	Parametre yardımcısı
CTRL+D	Tanımını bul
CTRL+G	Satıra git
CTRL+J	Seçili sözcüğün kod şablonunu yapıştır (varsa)
CTRL+<sayı>	İlgili numaralı yer imine git
CTRL+SHIFT+<sayı>	İlgili numaralı yer imini koy / kaldır
CTRL+SHIFT+I	Seçilmiş alanı daha içeri al
CTRL+SHIFT+U	Seçilmiş alanı daha dışarı al
CTRL+SHIFT+E	Projeyi düzenleme penceresini aç
CTRL+SHIFT+.	Yorum işareti ekleyerek yorum satırına dönüştür
CTRL+SHIFT+,	Yorum işaretini iptal et

CTRL+L	Fonksiyonlar penceresini göster
TAB	Seçili metni içeri (sağa) al
SHIFT+TAB	Seçili metni dışarı(sola) al
CTRL+SHIFT+I	Seçili metni içeri (sağa) al
CTRL+SHIFT+U	Seçili metni dışarı(sola) al
CTRL+ALT+Fare ile seç	Metni kolon olarak seç
ALT+Fare ile seç	Metni kolon olarak seç

Hata Ayıklayıcı (Debugger) kısayolları

F4	İmlece kadar çalıştır
F5	Durma noktasını ekle/kaldır
CTRL+SHIFT+F5	Tüm Durma-noktalarını sil
SHIFT+F4	Durma-noktaları penceresini göster
F6	Durma noktasına kadar çalıştır/beklet
F7	İlgili satırı çalıştır; satır fonksiyon ise içine gir (Step into)
F8	İlgili satırı çalıştır; satır fonksiyon ise içine girme (Step over)
CTRL+F8	Fonksiyondan çıkana kadar çalıştır (Step out)
F9	Hata ayıklayıcıyı çalıştır
CTRL+F2	Hata ayıklayıcıyı durdur
ALT+D	Assembly (sembolik makina kodu) karşılıklarını göster / gösterme

BÖLÜM 3



UYGULAMA YAPMA

mikroC'de uygulama yapmak kolay ve anlaşılırdır. Proje sihirbazı projelerinizi yalnızca birkaç tıklama ile oluşturmanızı sağlar: Uygulamanıza isim verme, yonga seçme, bayrak (flag) ayarı ve diğer parametreler gibi.

İsterseniz mikroC ile projelerinizi birden fazla kaynak dosyasına bölebilirsiniz. Bu sayede microC ile derlenmiş kütüphanenizi (.mcl dosyalarınızı) kaynak kodunuzu açıklamadan diğer kullanıcılarla paylaşabilirsiniz. En güzel özellik ise mikroPascal veya mikroBasic ile oluşturduğunuz .mcl dosyalarından mikroC'de yararlanabilmenizdir !

PROJELER

mikroC uygulamaları proje şeklinde düzenlenir; uygulama bir tek Proje dosyası (.ppc uzantılı) ve bir veya birden fazla kaynak dosyasına (.c uzantılı) sahip olabilir. Kaynak dosyalarını yalnızca bir projenin parçası olduğu takdirde derleyebilirsiniz. Proje dosyası aşağıdaki bilgileri taşır:
- projenin ismi ve opsiyonel olarak tarifi,
- hedef PIC MCU yonga,
- PIC ayar bayrakları (config word) ve yonga saat hızı,
- konuşlanma bilgileri ile birlikte projenin kaynak dosya listesi.

Yeni Proje


New Project.

Yeni bir proje oluşturmanın en kolay yolu proje sihirbazını kullanmaktır. Açılır menüden **Project > New Project** 'i seçin. Diyalog penceresindeki uygun yerleri (proje adı ve tanımı, yeri, kullanılan PIC yongası, saat hızı, config kelimesi) doldurun, böylelikle mikroC uygun proje dosyasını oluşturacaktır.

Ayrıca proje oluşturulduktan sonra isim verilecek boş bir kaynak dosyası yaratılır. mikroC sizden, kaynak dosyasına proje ile aynı ismi vermenizi zorunlu kılmaz, bu özellik sadece kullanım kolaylığı içindir.

Projeyi Biçimlendirme


Edit Project.

Daha sonra, açılır menüden **Project>Edit**'den projenizi düzenleyebilirsiniz. Projenize kaynak dosyaları ekleyip çıkarabilirsiniz, projeyi tekrardan isimlendirebilirsiniz, tanımları değiştirebilirsiniz, entegre (PIC MCU), saat, config word v.b. değiştirebilirsiniz. Kayıtlı herhangi bir projeyi silmek için sadece projenin kayıtlı olduğu klasörü silmek yeterlidir.

Projeye dosyaların eklenmesi/çıkarılması

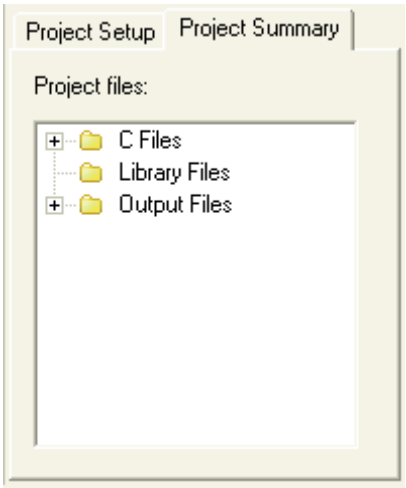

Add to Project.


Remove from Project.

Proje birkaç kaynak dosyası içerebilir (`.c uzantılı`). İlgili kaynak dosyalarının listesi proje dosyası içerisinde saklanır (`.ppc uzantılı`). Projenize kaynak dosyası ekleme işlemini; açılır menüden **Project > Add to Project** seçerek gerçekleştirebilirsiniz. Her eklenen kaynak dosyası kendi kendine yeterli olmalıdır, yani ön-işlemden sonra gerekli tüm tanımlamalara sahip olmalıdır. Dosya(ları) projenizden silme işlemini; açılır menüden **Project > Remove from Project** seçerek gerçekleştirebilirsiniz.
Not: Başlık dosyalarını eklemek için, #include ön işlem direktifini kullanınız.

Proje Dosyaları sekmesinin genişletilmiş özellikleri

Proje Dosyalarının yeni özelliklerini kullanırken, tek bir tıklama ile tüm çıkış dosyalarına (.lst, .asm) ulaşabileceksiniz. Aynı zamanda özel olarak kendi hazırladığınız veya derleyicinin hazır sunduğu kütüphane dosyalarını (.mcl türünde) proje içerisine dahil edebilirsiniz.



KAYNAK DOSYALARI

C kodlarını içeren kaynak dosyaları .c uzantılıdır. Uygulamaya ait kaynak dosyalarının listeleri de diğer bilgiler ile birlikte .ppc uzantılı proje dosyası içerisinde barındırılırlar. Kaynak dosyalarını sadece projenin bir parçası olduğu durumda derleyebilirsiniz.

Başlık dosyalarını (.h uzantılı) eklemek için `#include` ön-işlem (preprocessor) direktifini kullanabilirsiniz. Ön-işlemcinin diğer kaynak dosyalarını ekleyeceğine güvenmeyiniz - daha fazla bilgi için Projeler bölümüne bakın.

Arama Yolları

Kaynak dosyaları için arama yolları (Paths) (.c)

Kendi özel arama yollarınızı belirtebilirsiniz. Bunu açılır menüden önce **Tools > Options** daha sonra **Project > Search Paths'ı** seçerek gerçekleştirebilirsiniz.

Proje ayarları içerisinde, kaynak dosyaları için ya mutlak (yani kök dizinden itibaren) ya da göreceli arama yolu belirtebilirsiniz. Eğer göreceli arama yolunu belirtirseniz, mikroC aşağıdaki sıraya göre dosyayı arayacaktır:

1. proje dizini (.ppc proje dosyasını içeren dizin),
2. Sizin belirttiğiniz özel arama dizinleriniz,
3. mikroC kurulum dizini içindeki "Uses" alt dizini.

Başlık Dosyaları için yollar(.h)

Başlık dosyaları, `#include` ön-işlemci direktifi sayesinde dahil edilirler. Ön-işlemci direktifi içerisinde kaynak dosyasına tam bir yol yerleştirirseniz, sadece bu yer aranacaktır.

Kendi özel arama yolunuzu belirleyebilirsiniz: açılır menüden **Tools › Options** yolunu izleyerek **Search Path**'i seçiniz.

Proje ayarları içerisinde, başlık dosyaları için ya mutlak (yani kök dizinden itibaren) ya da göreceli arama yolu belirtebilirsiniz. Eğer göreceli arama yolunu belirtirseniz, mikroC aşağıdaki sıraya göre dosyayı arayacaktır:

1. proje dizini (.ppc proje dosyasını içeren dizin),
2. mikroC kurulum dizini içindeki "Uses" alt dizini,
3. Sizin belirttiğiniz özel arama dizinleriniz.

Kaynak Dosyalarının Yönetimi

Yeni bir kaynak dosyası oluşturma


New File.

Yeni bir kaynak dosyası oluşturmak için aşağıdaki aşamaları takip ediniz:

Açılır menüden **Select File > New** seçerek veya CTRL+N'ye basarak veya *New File* ikonuna tıklayarak yeni bir kaynak dosyası oluşturulur. Açılan dosya isim olarak "Untitled1" olarak açılacaktır. Bu sizin yeni kaynak dosyanızdır. Açılır menüden **Select File > Save As**'ı seçerek dosyanıza istediğiniz ismi verebilirsiniz.

Eğer, yeni proje oluşturma sihirbazını (New Project Wizard) kullanırsanız, daha sonradan isimlendirilecek .c uzantılı boş bir kaynak dosyası otomatik olarak oluşturulacaktır. mikroC kaynak dosyası isimlendirmesinin proje isimlendirmesiyle aynı olmasını şart koşmaz. Proje ile aynı veya farklı kaynak dosyası ismi seçmek tamamen keyfidir ve sizin seçiminize bırakılmıştır.


Open File Icon.

Varolan Bir Dosyayı Açma

Açılır menüden **Select File > Open** seçilerek veya CTRL+O'ya basılarak veya *Open file*'e tıklanarak *Select Input File* diyalog penceresi açılır. Dialog penceresinde, açmak istediğiniz dosyanın yerine gidiniz ve seçiniz, daha sonra *Open* butonuna tıklayınız.
Seçilen dosya kendi sekmesinde görüntülenecektir. Eğer seçilen dosya o anda açık ise, kendi bulunduğu Editör penceresi aktif hale gelecektir.


Print File Icon.

Açık Bir Dosyayı Yazdırmak

Yazdırmak istediğiniz dosyanın aktif olan (yani seçili) pencere olduğuna emin olun. Yazdırma işlemi için ya açılır menüden **Select File > Print** seçin veya CTRL+P'ye basın veya da *Print* ikonunu tıklayın. Yazdırma önizleme (Print Preview) penceresinde dokümanın istenen çıkış ayarlarını yapın ve OK butonuna basın. Böylece dosya istenen yazıcıda yazılacaktır.


Save File Icon.

Dosya Kaydetme

Aktif pencerede kaydetmek istediğiniz dosyanın olduğuna emin olun. Açılır menüden **Select File > Save** veya CTRL+S veya *Save* ikonu ile dosya kaydedilir. Dosya penceresindeki adı ile kaydedilecektir.


Save File As.

Dosyayı Başka Bir Ad Altında Kaydetmek

Aktif pencerede kaydetmek istediğiniz dosyanın olduğuna emin olun. Açılır menüden **Select File > Save As**'i seçin. Karşınıza *New File Name* diyalog penceresi açılır. Dialog penceresinde, dosyayı kaydetmek istediğiniz klasöre gidin. *File Name* penceresinde dosyaya vermek istediğiniz yeni adı yazın ve OK ikonuna basın. Dosya yeni adı ile kaydedilir.


Close File.

Bir Dosyayı Kapatma

Aktif pencerede kapatmak istediğiniz dosyanın olduğuna emin olun. Açılır menüden **Select File>Close'**a a tıklanarak veya *Code Editor*'ünün sekmesine sağ tıklanarak dosya kapatılabilir. Eğer en son kaydedilme işleminden sonra dosya değiştirilmişse mikroC geçerli değişikliklerin kaydedilip edilmeyeceğini sorar.

DERLEME (COMPILATION)

Compile Icon.

Projeyi oluşturup kaynak kodlarını yazdıktan sonra projeyi derlemek istersiniz. Bu işlem için açılır menüden **Project > Build**'i seçerek veya CTRL+F9'a tıklayarak veya build ikonuna tıklayarak kolayca derleme yapabilirsiniz.

İlerleme çubuğu (Progress bar) size derlemenin durumu hakkında bilgi vermek amacıyla ortaya çıkacaktır. Eğer hatalar varsa size *Error Window*'dan iletilecektir. Eğer hatayla karşılaşılmazsa mikroC çıkış dosyalarını üretecektir.

Çıkış Dosyaları

Sorunsuz bir derleme yapıldıktan sonra, mikroC çıkış dosyalarını proje klasöründe yaratacaktır (.ppc uzantılı proje dosyasını içeren klasör). Çıkış dosyaları aşağıda özetlenmiştir:

Intel HEX dosyası (.hex)
Intel formatında hex kayıtları dosyası; bu dosyayı PIC MCU'yu programlamak için kullanınız.

Binary derlenmiş mikro Kütüphanesi (.mcl)
Uygulamanın binary derlenmiş ve dağıtımı yapılabilir çeşidi; diğer projelere dahil edilebilir.

Listeleme Dosyası (.lst)
PIC'in bellek ayırımının listesini verir; komut adreslerini, kayıtçıları, yordamları vs. gösterir.

Sembolik Makina Dili Dosyası (.asm)
List File'dan alınmış sembolik isimleri de içeren ve okunabilir formatta olan bir sembolik makina dili dosyasıdır.

View Assembly Icon.

Sembolik Makina Dili (Assembly) Görünümü

Programınızı mikroC'de derledikten sonra, *View Assembly* ikonuna tıklayarak veya açılır menüden **Project > View Assembly** 'i seçerek oluşturulmuş Sembolik Makina dili kod dosyasına (.asm) yeni bir pencereden ulaşabilirsiniz. Sembolik Makina dili, PIC MCU komutlarının sembolik isimlerle okunabilir halidir. Tüm fiziksel adresler ve diğer bilgiler istatistik penceresinde veya listeleme dosyasının içerisinde bulunabilir (.lst).

HATA MESAJLARI

Hata Mesajları

- Specifier needed	: Belirtece gerek var
- Invalid declarator	: Geçersiz bildirici
- Expected '(' or identifier	: Beklenen '(' veya tanıtıcı
- Integer const expected	: Tamsayı sabiti bekleniyor
- Array dimension must be greater than 0	: Dizi boyutu 0'dan büyük olmalı
- Local objects cannot be extern	: Yerel nesneler harici tanımlanmış olamaz
- Declarator error	: Bildirici hatası
- Bad storage class	: Uygunsuz depolama sınıfı
- Arguments can not be of void type	: Fonksiyon bağımsız değişkenleri void tipte olamaz
- Specifier/qualifier list expected	: Belirteç/niteleyici listesi bekleniyor
- Address must be greater than 0	: Adres 0'dan büyük olmalı
- Identifier redefined	: Tanıtıcı tekrar tanımlanmış
- case out of switch	: "switch" kalıbı dışında "case" anahtar kelimesi kullanılmış
- default label out of switch	: "switch" kalıbı dışında "default" anahtar kelimesi kullanılmış
- switch exp. must evaluate to integral type	: switch kalıbının sonucu tamsayı tipinde olmalı
- continue outside of loop	: Döngü dışında "continue" anahtar kelimesi kullanılmış
- break outside of loop	: Döngü dışında "break" anahtar kelimesi kullanılmış
- void function cannot return values	: void fonksiyonu değer geri döndüremez
- Unreachable code	: Ulaşılamaz kod

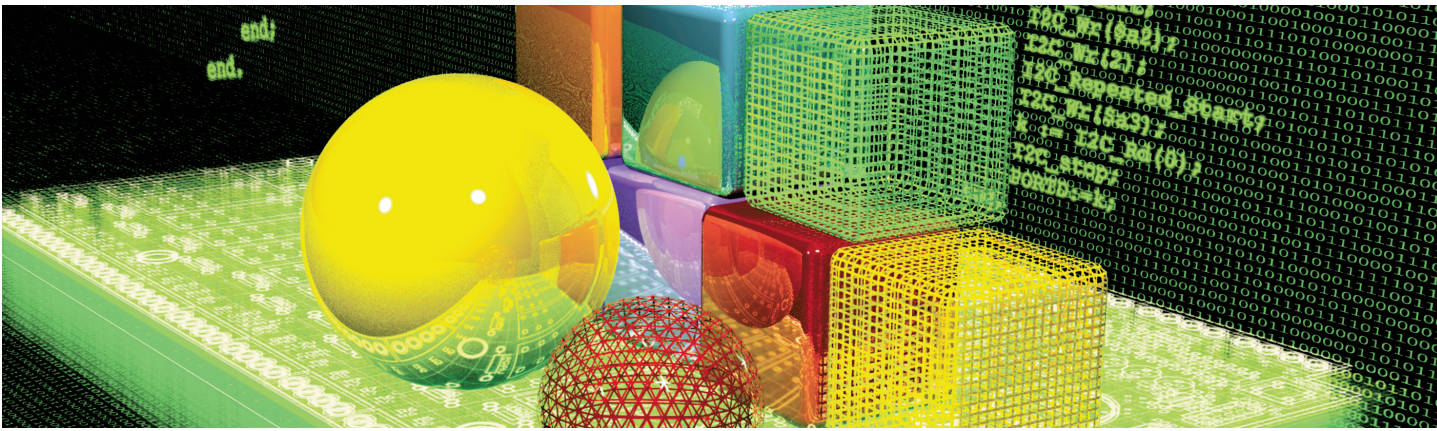
- Illegal expression with void	: "void" kullanılan geçersiz ifade
- Left operand must be pointer	: Sol tarafta işlenen eleman işaretçi olmalıdır
- Function required	: İşlev gerekli
- Too many chars	: Çok fazla karakter
- Undefined struct	: Tanımlanmamış yapı
- Nonexistent field	: Var olmayan alan
- Aggregate init error	: Yığın ilk değeri verme hatası
- Incompatible types	: Birbirine uyuşmayan tipler
- Function definition not found	: Fonksiyon tanımı bulunamadı
- Signature does not match	: Kalıp uyuşmazlığı
- Cannot generate code for expression	: İfade için kod üretilemedi
- Too many initializers for subaggregate	: Çok fazla alt-yığın ilk değeri
- Nonexistent subaggregate	: Mevcut olmayan alt-yığın
- Stack overflow : func call in complex expression	: Yığıt aşımı: karmaşık ifade içerisinde işlev dallanması
- Syntax Error: expected %s but %s found	: Sözdizimi hatası: beklenen %s fakat %s bulundu
- Array element cannot be function	: Dizi elemanı işlev olamaz
- Function cannot return array	: İşlev diziyi geri döndüremez
- Inconsistent storage class	: Tutarsız depolama belleği sınıfı
- Inconsistent type	: Tutarsız tip
- %s tag redefined	: %s etiketi tekrar tanımlanmış
- Illegal typecast	: Geçerli olmayan tip çevirimi
- %s is not a valid identifier	: %s geçerli bir tanıtıcı değil

- Invalid statement : Geçersiz deyim
- Constant expression required : Sabit ifade gerekli
- Internal error %s : İçsel hata %s
- Too many arguments : İşlevde çok fazla bağımsız değişken
- Not enough parameters : Yetersiz sayıda parametre
- Invalid expression : Geçersiz ifade
- Identifier expected, but %s found : Tanıtıcı beklendi, fakat %s bulundu
- Operator [%s] not applicable to this operands [%s] : [%s] işleyicisi (operator), [%s] işlenen (operand) için uygulanabilir değil
- Assigning to non-lvalue [%s] : lvalue olmayan yani atama yapılamayan [%s] 'a atama
- Cannot cast [%s] to [%s] : [%s], [%s] 'ya çevrilemiyor
- Cannot assign [%s] to [%s] : [%s], [%s] 'ya atanamıyor
- lvalue required : lvalue yani atanabilen eleman gerekli
- Pointer required : İşaretçi gerekli
- Argument is out of range : Argüman erimin dışında
- Undeclared identifier [%s] in expression : İfade içerisinde daha önceden belirtilmemiş [%s] tanımlayıcısı
- Too many initializers : Çok fazla ilk değer atayıcı
- Cannot establish this baud rate at %s Mhz clock : %s MHz clock'ta bu baud oranı kurulamaz

Derleyici Uyarı Mesajları

- Highly inefficient code: func call in complex expression : Çok verimsiz kod: karmaşık ifade içinde işlev (fonksiyon) çağrısı
- Inefficient code : func call in complex expression : Verimsiz kod: karmaşık ifade içinde işlev (fonksiyon) çağrısı

BÖLÜM 4



mikroC Dil Referansı

Neden C öncelikli? Cevabı çok basit: okunaklıdır, öğrenilmesi kolaydır, yapısal bir programlama dilidir ve mikrodenetleyici programlama için yeterli güç ve esnekliğe sahiptir. Daha önceden programlama tecrübeniz olmasa bile mikroC ile program yazmanın çok kolay bir çalışma olduğunu göreceksiniz. Bu bölüm PIC mikro denetleyicilerini nasıl programlayacağınıza, ayrıca C ile program yazma kurallarını öğrenmenize ve/veya hatırlamanıza yardımcı olacaktır.

PIC MİKRODENETLEYİCİSİNE ÖZEL

mikroC derleyicisinden daha fazla faydalanabilmek için, PIC MCU'nun temel özelliklerini bilmek gerekmektedir. Bu bilgi zorunlu değildir, fakat PIC'in tüm yeteneklerini ve kısıtlamalarını ve bunların kod yazmadaki etkinliklerini daha iyi anlayabilmenizi sağlayabilir.

Tip Verimliliği

Herşeyden önce PIC'in ALU'sunun byte düzeninde aritmetik işlemleri yapmak için optimize edilmiş olduğunu bilmek gerekir. mikroC çok karmaşık verileri işleyebilse de, PIC MCU; özellikle eski tipler, bu kapasiteyi sınırlayabilir. Bu da çok basit uygulamalar için bile fazla zaman harcanmasına yol açabilir. Evrensel tavsiye her uygulama için mümkün olan en küçük veri tipini kullanmaktır. Bu tavsiye her türlü programlama için geçerlidir ve PIC MCU'lar için daha önemlidir.

Hesaplamaya gelindiğinde tüm PIC'ler aynı performansa sahip değildir. Örnek olarak, PIC16 ailesi iki byte'ı çarpmak için gerekli donanıma sahip değildir, bu nedenle çarpma işlemini bir yazılım algoritması ile gerçekleştirir. Diğer taraftan, PIC18 ailesi donanımsal çarpma devresine sahiptir ve sonuç olarak çarpma işlemini PIC16 ailesine göre daha hızlı tamamlar.

İç içe Dallanmaların Kısıtlamaları

İç içe dallanma bir fonksiyon içinde yine aynı fonksiyonun (özyineli çağırma) veya başka bir fonksiyonun çağırılması işlemidir. Özyineli (recursive) çağırma mikroC tarafından sınırlı olarak desteklenir. PIC MCU'nun yığıt (stack) ve bellek sınırlamalarından dolayı özyineli çağırmalar lokal değişkenler ve fonksiyon parametreleri içeremezler.

mikroC özyineli olmayan iç içe fonksiyon çağırılma sayısını şu şekilde kısıtlar:

- PIC12 ailesi için 8 çağırma,
- PIC16 ailesi için 8 çağırma,
- PIC18 ailesi için 31 çağırma geçerlidir.

İzin verilen içiçe fonksiyon çağırma sayısı, kod içerisinde * / % operatörlerinden birinin kullanılması durumunda bir (1) azalır. Program içinde kesme kullanılması durumunda bir (1) daha azalır. Yeniden girişli (reentrant) fonksiyonlar da sınırlı şekilde (parametre ve lokal değişken olmadığında) desteklenmektedir. Eğer izin verilen çağırma sayısı aşılırsa, derleyici "Yığıt Aşım Hatası" verecektir.

Sadece PIC16'ya Özgü Özellikler

Sayfa Bölme

PIC16'da tek bir yordam bir kod sayfasını (2.000 komut) geçemez. Eğer yordam bir kod sayfasına sığmazsa bağlayıcı (linker) hata raporu verecektir. Böyle bir problemle karşılaşıldığında uygulamanızı tekrar tasarlamayı düşünmelisiniz. Örneğin ilgili yordamı birkaç parçaya bölmek bir çözüm olabilir.

FSR Üzerinden Dolaylı Yaklaşım Limitleri

PIC16'nın işaretleyicileri "yakındır", yani bunlar adresin sadece alt 8-bitini taşırlar. Derleyici başlangıçta 9'uncu biti otomatik olarak siler, bu durumda işaretleyiciler sadece 0'ıncı ve 1'inci bloğu işaretler. 3'üncü ve 4'üncü bloktaki nesnelere ulaşmak için kullanıcı STATUS yazmacının IRP bitini manuel olarak 1 yapmalı ve işlemini tamamladıktan sonra yine 0 yapmalıdır. Belirtilmiş kurallar herhangi bir dolaylı yaklaşıma uygulanır: diziler, yapılar ve ortaklık (union) atamaları... vs

Not: Bu metodu kullanıyorsanız, IRP bit alanını düzgün kullanmak çok önemlidir. Eğer birçok değişkeniniz varsa, dolayısıyla bu metod sizin için uygun ve pratik değil ise PIC16 yerine PIC18 kullanımına geçmeniz gerekir.

Not: Eğer kod içerisinde birçok değişkene sahipseniz, bunları bağlayıcının (linker) "absolute" direktifi ile yeniden düzenleyiniz. Programın verimini arttırmak için, sadece direkt olarak ulaşılan ve adreslenen değişkenler 3'üncü ve 4'üncü bloğa taşınmalıdır .

mikroC'YE ÖZEL

ANSI Standardı Konusunda

ANSI C Standardından Sapmalar

mikroC, birkaç alanda ANSI C'den ayrılır. Bu değişikliklerin bazıları PIC programlamayı kolaylaştırma amaçlıdır, bazıları ise PIC mikrodenetleyicilerinin donanım sınırlamaları sonucu ortaya çıkmıştır :

Fonksiyon özyineleme (recursion), kolay kullanımı olmayan yığıt (stack) ve sınırlı bellekten dolayı sınırlı olarak desteklenmektedir.

Değişken işaretçileri ve sabit işaretçileri birbirleri ile uyumlu değildir, yani herhangi bir atama ve karşılaştırma ikisi arasında mümkün değildir.

mikroC, const niteleyicisi ile tanıtılmış tanıtıcıları "gerçek sabit" olarak kabul eder (C++ stili). Bu ANSI C'nin sabit bir ifadeyi mecbur koştuğu yerlerde sabit nesnelerin kullanılmasına olanak sağlar. Ayrıca eğer kod taşınabilirliği amacınız var ise geleneksel ön-işlemci tanımlı sabitleri kullanın.Tip Nitelemeleri ve Sabitlere bakınız.

mikroC, iki slaş işareti(//) ile kullanılmış tek satırlık C++ stili yorumların kullanılmasına izin verir.

Uygulama-Tanımlı Davranış

ANSI standardının bazı bölümleri uygulama-tanımlı davranışa sahiptirler. Bu şu anlama gelmektedir, bazı C kodlarının tam davranışları derleyiciden derleyiciye değişmektedir. Yardım (help) bölümünde mikroC derleyicisinin bu tür durumlarda nasıl davranacağı anlatılmıştır. En önemli özellikler şunlardır: Kayan-noktalı (floating-points) Tipler, Depolama (Storage) Sınıfları ve Bit Alanlarıdır.

Önceden Tanımlı Değişkenler ve Sabitler

Programlamayı kolaylaştırmak için, mikroC önceden tanımlanmış birtakım global değişken ve sabitler(Constants) içerir.

PIC'in tüm SFR yazmaçları `volatile unsigned short` global değişkenleri olarak tanımlıdırlar ve tüm projede görünürler. Bir proje oluşturulduğunda, mikroC geçerli SRF'leri ve sabitleri (PORTB, TMR1 gibi) içeren uygun bir .def dosyasını kapsamına alacaktır. Belirteçler, Microchip'in veri sayfalarında standart olduğu gibi büyük harflidirler. Tüm tanımlı değişkenler ve sabitler için mikroC'in kurulum klasöründeki "defs" klasörüne bakınız ya da *Kod Asistanı*'nı inceleyiniz (Editörün içerisinde: CTRL+SPACE).

Tek-tek Bitlere Erişim

mikroC size, `char` ve `unsigned short` tiplerinin değişkenlerinin 8 bitine de tek-tek erişebilme imkanı sağlar. Bu işlem için basitçe bir değişkenle birlikte eleman seçici olan noktayı (.) ve takibinde de F0,F1,...,F7 tanıtıcılarından birini kullanınız. Örnek olarak:

```
// Eğer RB0 set edilmişse, RC0'ı set et:
if (PORTB.F0) PORTC.F0 = 1;
```

Burada herhangi bir özel tanıma gerek yoktur; bu tür seçmeli erişimler mikroC'in içerisinde yerleşik haldedir ve kodun her yerinde kullanılabilir. F0-F7 tanımlayıcıları büyük-küçük harf duyarlı değildirler ve özel bir isim-havuzu içerisindedirler.

Eğer kullandığınız PIC MCU'ya aşina iseniz bitlere, bitlerin isimleri ile de ulaşabilirsiniz:

```
INTCON.TMR0F = 0;  //  TMR0F' yi temizle
```

Yazmaçlar ve bit isimleri hakkında daha detaylı bilgi için, Önceden Tanımlı Değişkenler ve Sabitler bölümüne bakın.

Not: Eğer amaç taşınabilirlik ise, bu tip tek-tek bitlere erişimi kullanmayın, onun yerine bit alanlarını (field) kullanın.

Kesmeler (Interrupts)

Kesmeler (interrupt), özellikle rezerve edilmiş *interrupt* (kesme) kelimesi ile kolayca kullanılabilirler. `interrupt` işlevi mikroC tarafından zaten tanımlanmış olup kullanıcı tarafından tekrardan tanımlanamaz. Prototipi:

```
void interrupt(void);
```

Uygulamanız içerisindeki kesme işlemi için kendi kesme işlev tanımınızı (işlev gövdesini) yazınız. mikroC kesme alt-yordamına girerken aşağıdaki SFR'leri MCU yığıtına (stack) kaydeder, kesme alt-yordamından dönüşte bu yazmaçlara tekrar yığıtdaki eski değerlerini yükler:
PIC12 ve PIC16 ailesi için: `W, STATUS, FSR, PCLATH.`
PIC18 ailesi için: `FSR.` (`WREG, STATUS` ve `BSR`'yi saklamak için fast context özelliği kullanılır.)

Not: PIC18 ailesi için yüksek öncelikli kesmeler yukarıdaki ile aynı anahtar sözcük ile ifade edilir. Düşük öncelikli kesmeler için ise interrupt_low anahtar sözcüğü kullanılır. Prototipi:

```
void interrupt_low(void);
```

Kesme (interrupt) İçerisinden İşlev (Fonksiyon) Çağırma
Kesme yordamından işlev (fonksiyon) çağırmak artık mümkündür. Derleyici, hem "ana-programda" ve hem de "kesme alt yordamında" kullanılan yazmaçlara dikkat eder ve akıllı durum saklama yöntemi ile sadece her iki yordamda da kullanılan yazmaç içeriklerini saklar. Detaylar için **Fonksiyonların yeniden-giriş '**i bölümüne bakınız.

Burada TMR0'dan kaynaklanan bir kesmenin nasıl işlendiği gösterilmiştir (Tabi eğer başka bir kesme kullanılmasına izin verilmemiş ise):

```
void interrupt() {
  counter++;
  TMR0 = 96;
  INTCON = $20;
} //~
```

Çoklu kesmeye izin verilmiş olması durumunda, hangi kesmelerin oluştuğunu test etmeniz gerekmektedir ve sonrasında da uygun kod ile durumu ele almalısınız. (kesme 'interrupt' işleme)

Bağlayıcı (Linker) Direktifleri

mikroC nesneleri bellekte dağıtmak için iç algoritma kullanır. Eğer önceden belirtilmiş bir adreste değişken veya yordama ihtiyacınız varsa `absolute` ve `org` bağlayıcı direktiflerini kullanabilirsiniz.

Absolute Direktifi

`absolute` direktifi, değişkenlerin RAM içerisindeki başlama adreslerini gösterir. Eğer değişken çok byte'lı ise, yüksek byte'lar ardışık adreslerde depolanır. `absolute` direktifi değişkenlerin tanımlarının sonuna eklenir:

```
int foo absolute 0x23;
// değişken 0x23 ve 0x24 adreslerinde 2 byte yer kaplayacak.
```

`absolute` direktifini kullanırken çok dikkatli olmak gerekir. Örneğin iki değişken yanlışlıkla üstüste çakışarak aynı adrese yazılabilir. Mesela:

```
char i absolute 0x33;
// i değişkeni 0x33 adresinde 1 byte yer kaplayacak.

long jjjj absolute 0x30;
// değişken 0x30, 0x31, 0x32, 0x33 adreslerinde 4 byte yer
// kaplayacak, bu nedenle
// i'nin değişimi jjjj'nin en üst byte'ının da
// değişmesine sebep olacaktır.
```

org Direktifi

`org` direktifi, yordamların ROM'daki başlama adreslerini belirtir.

`org` direktifi yordamların tanımlarının sonuna konur. Belirli olmayan tanımlamalara uygulanan direktifler iptal edileceklerdir, bağlayıcı (linker) tarafından uygun bir uyarı yapılacaktır. `org` direktifi bir kesme yordamına uygulanamaz.

Örnek:

```
void func(char par) org 0x200 {
// fonksiyon 0x200 adresinde başlayacaktır
  nop;
}
```

Kod Optimizasyonu

Optimizer derleyicinin kullanılabilirliğini arttırmak için eklenmiştir. Oluşturulan kodun uzunluğunu kısaltır, dolayısıyla kodun çalışma hızını arttırır. Ana özellikleri şunlardır:

Sabit Kısaltımı
Derleme anında hesaplanabilen tüm ifadeler (mesela: sabitler) kendi sonuçları ile değiştirilirler: (3+5-->8)

Sabit Yayılımı
Belirli bir değişkene bir sabit atandığı zaman derleyici bunu tanır ve değişkenin değeri değiştirilmediği sürece, değişken bu sabitle değiştirilir.

Kopya Yayılımı
Derleyici, aynı değere sahip iki değişken bulursa kodun devamında birini eler.

Değer Numaralandırma
Eğer derleyici, aynı sonucu veren iki ifade bulursa, kodun devamında hesaplamalardan birini eler.

Ölü Kod (Dead Code) Eleme
Kod içerisinde başka hiçbir yerde kullanılmayan ve sonucu etkilemeyen “kod parçaları” otomatik olarak silinir.

Yığıt Ayırımı
Geçici yazmaçlar (“Yığıt”, “Stack”) çok hesaplı kullanılır ve böylece karışık ifadeler asgari yığın kullanımı ile gerçekleştirilir.

Yerel Değişken Optimizasyonu
Sonuçları bazı global veya geçici (volatile) değişkenlere etki etmeyen yerel değişkenler kullanılmaz.

Daha İyi Kod Üretimi ve Yerel Optimizasyon
mikroC derleyicisinde kod geliştirme işlemi çok tutarlıdır ve kod uzunluğunu kısaltmak için ”kod yapı taşları” geliştirilmesine özen gösterilmiştir.

Dolaylı Fonksiyon Çağırımları

Eğer bağlayıcı dolaylı bir fonksiyon çağırması ile karşılaşırsa (fonksiyon işaretçisi sayesinde), programın herhangi bir yerinde adresleri alınmış olan fonksiyonlardan herhangi birinin bu noktada çağrılabileceğini kabul eder. Mevcut fonksiyondan hangi fonksiyonların dolaylı olarak çağrılabileceğini bildirebilmek için `#pragma funcall` direktifini kullanınız:

#pragma funcall <fonk_ismi> <cagrilan_fonk>[, <cagrilan_fonk>,...]

`fonk_ismi` fonksiyonunun gerçeklendiği kaynak modüle uygun pragma yerleştirilmelidir. Ayrıca bu modül `cagrilan_fonk` listesindeki tüm listelenmiş fonksiyonların bildirimini de (declaration) içermelidir .

`fonk_ismi`'nin kod içerisinde kullanılması halinde ise `cagrilan_fonk` listesindeki tüm fonksiyonlar kullanılıp kullanılmadıklaırna bakılmaksızın bağlayıcı tarafından bağlanacaklardır.

Not: `#pragma funcall` direktifi bağlayıcıya (linker), derlenen yığıt içerisindeki fonksiyon çerçevesinin yerleşiminin optimize edilmesine yardım edebilir.

SÖZLÜKSEL ÖĞELER (LEXICAL ELEMENTS)

Bu başlık mikroC'in sözlüksel elemanlarının formal tanımını vermektedir. Sözlüksel elemanlar bir programlama dili tarafından tanınan kelime benzeri yapıların (dizgecik, token) farklı sınıflandırılmasını anlatır.

Derleme işleminin dizgeleme evresinde kaynak kod "dizgecikler" ve "beyaz uzay" olmak üzere ayrıştırılır. mikroC'de dizgecikler önişlemci ve derleyicinin kullanıcı programı üzerinde yaptığı bir kısım işlem ile ayrıştırılır.

mikroC programı, uygun bir metin editörü (örneğin: mikroC Kod Editörü) ile yazılmış bir dizi ASCII karakterden oluşan kaynak kodu yazımı ile başlar. mikroC'de en temel programlama ünitesi bir dosyadır. Bu dosya, genelde RAM veya disk üzerinde ismi belli bir dosya olup dosya uzantısı .c'dir.

Beyaz Uzay (Whitespace)

Beyaz Uzay (Whitespace), boşluklara (blanks), yatay ve dikey sekmelere verilmiş genel bir isimdir. MikroC'de kaynak koddaki yorumlar da beyaz uzay olarak kabul edilir. Whitespace program dizgeciklerinin (tokenlerin) nerede başladığını ve nerede bittiğini bulmada yararlıdır. Bu işlevin ötesinde, kod içerisinde gereksiz beyaz uzay varsa bunlar gözönüne alınmaz. Örnek olarak aşağıdaki iki dizi:

```
int i; float f;
```

ve

```
int i;
    float f;
```

sözlüksel olarak eşittirler ve aynı şekilde altı dizgecik olarak ayrıştırılırlar.

Sabit diziler içerisinde özellikle beyaz uzay göstermek gerekebilir, bu durumda bu kısımlar ayrıştırma işleminden muaf tutulurlar ve böylece karakter dizisinin bir parçası olarak kalırlar.

Yorumlar (Comments)

Yorumlar, bir programın anlaşılabilirliğini arttıran ek notlar, metin parçalarıdır ve teknik olarak başka bir beyaz uzay çeşididir. Yorumlar sadece programcının kullanımı içindir; derleyici ayraç işlevinden önce yorumları kaynak kod metninden ayrıştırır. Yorumları tanımlamada iki yol mevcuttur: C metodu ve C++ metodu. Her ikiside mikroC'de desteklenmektedir.

C tipi yorumlar

C yorumları /* sembol çiftinden sonra bırakılmış herhangi bir karakter dizisidir. Yorum başlangıç /* tan sonraki ilk */ ile sonlandırılır. Tüm dizi, yani yorum için kullanılan dört sembol ve aradaki tüm satırlar yorum olarak kabul edilir. Yorumlar bir boşluk karakteri olarak kabul edilirler.

mikroC'de,

```
int /* tip */ i /* tanimlayici */;
```

şöyle ayrıştırılır:

```
int i;
```

Unutmayın ki mikroC taşınabilir bir dizgecik yapıştırma yöntemi olmayan /**/ kullanımını desteklemez. Simge yapıştırması hakkında daha fazla bilgi için Önişlemci bölümüne bakınız.

C++ tipi yorumlar

mikroC çift slaş (//) kullanarak oluşturulan tek satırlık yorumları destekler. Yorum herhangi bir kolondan başlayabilir ancak satırın sonuna kadar devam edebilir:

```
int i;  // bu tumce bir yorumdur
int j;
```

mikroC tarafından şöyle kabul edilerek ayrıştırılır:

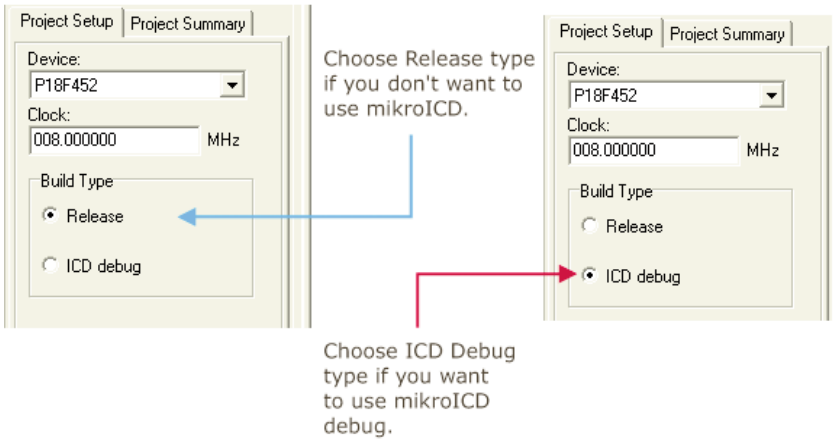
```
int i;
int j;
```

mikro ICD - Devre Üzerinde Hata Ayıklayıcı (In-Circuit Debugger)

mikro ICD donanım seviyesinde gerçek zamanlı hata ayıklamada çok etkili bir gereçtir. ICD hata ayıklayıcı bir mikroC programını bir host PIC mikrodenetleyici üzerinde koşturmanızı sağlar ve program çalışırken; değişkenlerin değerlerini, Özel Görevli Yazmaçları (SFR), bellek ve EEPROM durumunu gösterir.

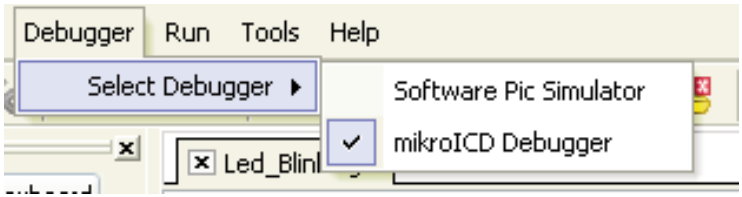
Step No. 1

mikro ICD'yi kullanmak için uygun donanım ve yazılıma sahipseniz programını tamamladığınızda Build type olarak **Release** veya **ICD debug**'dan birini seçebilirsiniz.



Step No. 2

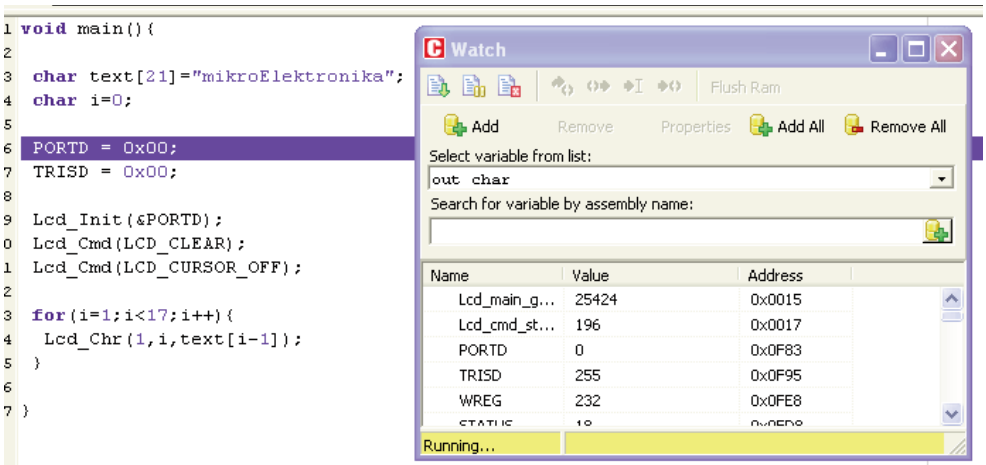
ICD Debug build tipini seçtikten sonra projenizin derleme işlemine geçebilirsiniz. Projenizi başarı ile derledikten sonra F11 kısayolu ile PIC Mikrodenetleyiciyi programlayınız. Daha sonra açılır menüden **Debugger > Select Debugger > mikroICD Debugger**'ı seçerek hata ayıklama birimini başlatınız.



Step No. 3

mikro ICD'yi ya açılır menüden **Run > Debug**'ı seçerek ya da basitçe *Debug* ikonuna tıklayarak çalıştırabilirsiniz. Hata ayıklayıcıyı başlattığınızda önünüze daha fazla seçenekler gelecektir: "İçine gir" (Step Into), "Üstünden atla" (Step Over), "İmlece kadar çalıştır" v.b.

Çalıştırılacak olan komut satırı renklendirilir (Önayar rengi mavidir). Ayrıca programın çalışması ile ilgili bilgiler Gözlem Penceresinde (**Watch Window)** sarı ile gösterilen "Durum" (Status) satırında verilir. Bazı fonksiyonların işlenmesi zaman aldığından bu sırada Gözlem Penceresinde "Program koşuyor..." (Running...) yazısı yazılır.



Faydalı bilgi : Hata ayıklayıcıya özel
Açıklama : Derleme tipini Release'da seçseniz, ICD Debug'da seçseniz her iki durumda da "Software Pic Simulator" yani simülasyon modunda çalışabilirsiniz. (Sadece derleme sonucundaki koşturulabilir dosyalarınızın boyutu farklı olur.) Ancak mikroICD Debugger ile PIC üzerinde hata ayıklamak isterseniz mutlaka ICD Debug seçeneği ile derleme yapmanız gereklidir. (Ve böylece hata ayıklamaya uygun koşturulabilir dosya üretmiş olursunuz.)

mikro ICD Hata Ayıklayıcı Seçenekleri

İsim	Tanım	Fonksiyon tuşu
Debug	Hata ayıklayıcıyı başlatır.	**[F9]**
Run/Pause Debugger	Hata ayıklayıcıyı Başlatır / Duraklatır.	**[F6]**
Toggle Breakpoints	İmleç pozisyonunda durma noktasını koyar/kaldırır. Tüm durma noktalarını görüntülemek için açılır menüden **Run > View Breakpoints**'i seçiniz.	**[F5]**
Run to Cursor	Mevcut komut ile imlecin bulunduğu satır arasındaki komutları işler.	**[F4]**
Step Into	Mevcut C komutunu (tek veya çok çevrimli) işler ve durur. Eğer komut bir yordam çağırma ise yordamın içine girer ve çağırma komutundan sonraki ilk komutta durur.	**[F7]**
Step Over	Geçerli komutu çalıştırır, ardından durur. Eğer komut bir yordam dallanması ise, yordama girmez ve devam eden ilk komutta durur.	**[F8]**
Flush Ram	PIC belleğini temizler, tüm değerler gözlem penceresindeki değerler ile değiştirilir.	**N/A**

mikro ICD Debugger Örneği

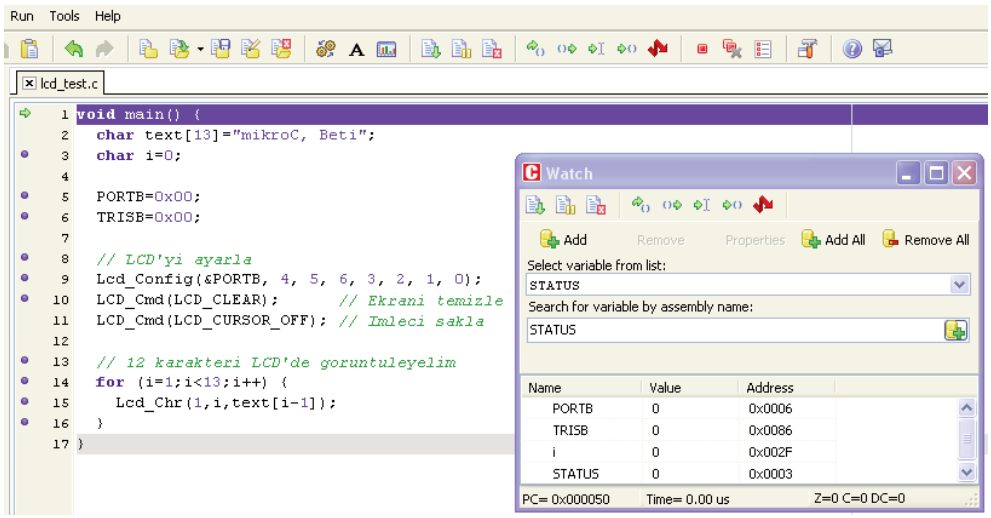
Step No. 1

Burada mikro ICD ile hata ayıklamanın nasıl yapıldığını adım adım göstereceğiz. Tabii öncelikle bir program yazmanız lazım. Biz aşağıdaki program üzerinde mikroICD'nin nasıl çalıştığını göstereceğiz:

```
void main(){
  char text[13]="mikroC, Beti";
  char i=0;

  PORTB=0x00;
  TRISB=0x00;

  Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0); // LCD'yi ayarla
  LCD_Cmd(LCD_CLEAR);       // Ekrani temizle
  LCD_Cmd(LCD_CURSOR_OFF); // Imleci sakla

  // 12 karakteri LCD'de goruntuleyelim
  for (i=1;i<13;i++) {
    Lcd_Chr(1,i,text[i-1]);
  }
}
```

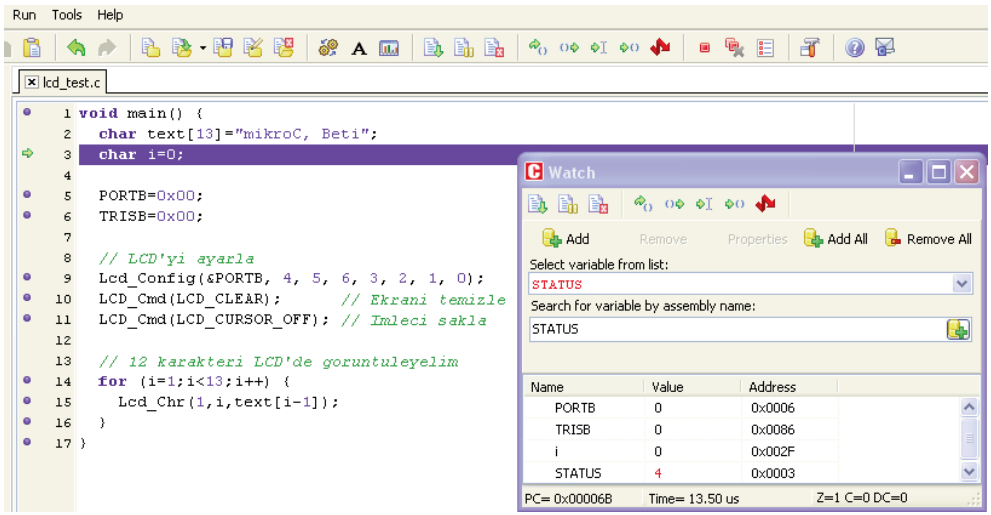
Step No. 2

Başarılı bir derleme ve PIC'i programlamadan sonra mikro ICD'yi başlatmak için **F9** tuşuna basınız. mikroICD'nin başlatılmasından sonra mavi aktif satır gözükecektir.

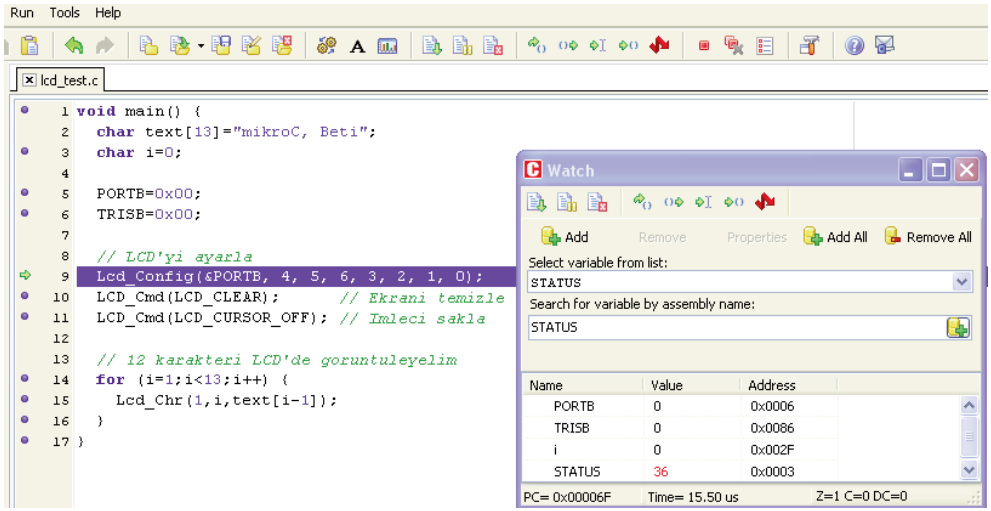


Step No. 3

Şimdi programda satır satır hata ayıklayacağız. **F8** tuşuna basmak programı satır satır işletecektir. Gecikme yordamlarında veya gecikme içeren diğer yordamlarda kullanıcı *Step Into* **[F7]** ve *Step Over* **[F8]** komutlarını kullanmamalıdır. Bunların yerine *Run to Cursor* **[F4]** veya durma noktası (breakpoint) fonksiyonlarını kullanmalıdır.

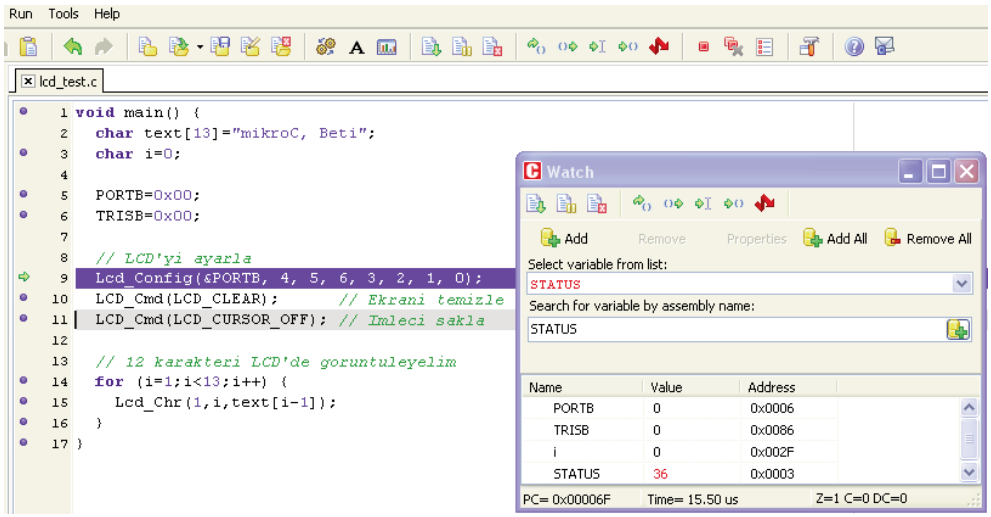


Bütün değişiklikler PIC'den okunacak ve Gözlem Penceresine (Watch Window) yüklenecektir. Dikkat ederseniz **STATUS** Özel Fonksiyonlu Yazmacı değişmiştir.



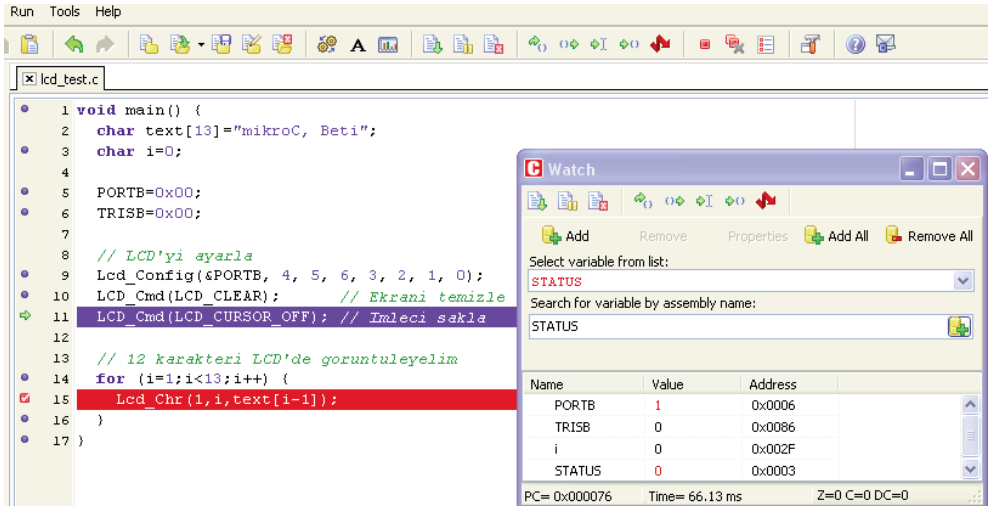
Step No. 4

Step Into **[F7]** ve *Step Over* **[F8]** komutları adım modunda kullanılan mikroICD fonksiyonlarıdır. Ayrıca mikroICD tarafından desteklenen Gerçek-Zaman (Real-Time) modu vardır ki bu modda *Run/Pause Debugger* **[F6]** ve *Run to Cursor* **[F4]** fonksiyonları kullanılır. **[F4]**'e basmak kullanıcı tarafından seçilen satıra gitmektir. Kullanıcı satırı seçip **F4**'e basmalıdır. Bu durumda kod seçilen satıra ulaşana kadar işlenir.

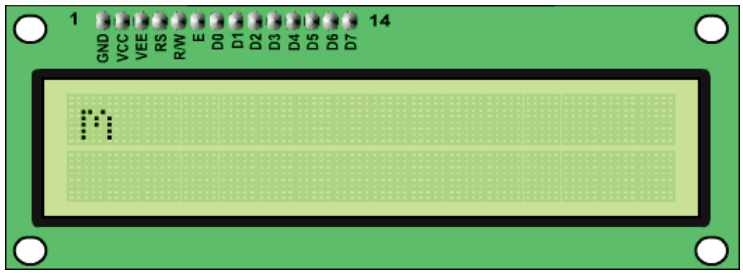


Step No. 5

Run/Pause Debugger **[F6]** ve *Toggle Breakpoints* **[F5]** Gerçek-Zaman modunda kullanılan mikro ICD hata ayıklama fonksiyonlarıdır. **F5**'e basılınca kullanıcının seçtiği satıra durma noktası konur. **F6** tuşu kodu, durma noktasına gelene kadar çalıştırır. Durma noktasına ulaşınca Hata-Ayıklayıcı durur. Buradaki örneğimizde durma noktalarını LCD'ye karakter karakter "mikroC, Beti" yazmak için kullanacağız. Durma noktası LCD_Chr üzerine konmuştur, dolayısıyla program bu fonksiyona her ulaştığında duracaktır. Durma noktasına ulaşıldığında programı yeniden başlatmak için **F6** tuşuna basılmalıdır.



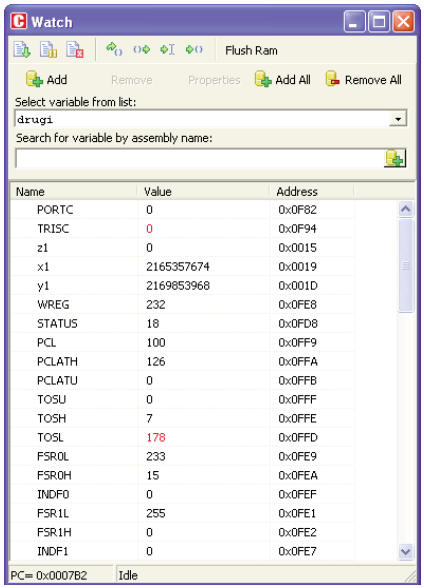
Durma noktaları iki gruba ayrılır; donanımsal ve yazılımsal durma noktaları. Donanımsal durma noktaları PIC üzerindedir ve en hızlı hata ayıklama işine yarar. Donanımsal durma noktası sayısı limitlidir (P16 serisi için 1, P18 serisi için 1-3). Eğer tüm donanımsal durma noktaları kullanılmış ise bir sonra kullanılacak durma noktası yazılımsal olmalıdır. Yazılımsal durma noktaları mikro ICD'nin içindedir ve donanımsal durma noktasına benzetim yaparlar. Ancak yazılımsal durma noktaları donanımsal durma noktalarından çok daha yavaştır. mikro ICD içerisinde iki tür durma noktası sadece hız yönünden fark edilebilir. Bu nedenle iki tür durma noktası olduğunu iyi bilmek gerekmektedir.



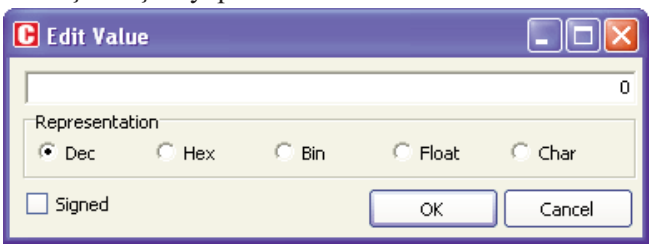
mikro ICD (Devre Üzerinde Hata Ayıklayıcı) Özet

Gözlem Penceresi (Watch Window)

Gözlem Penceresi (Watch Window) programın akışı sırasında tüm program bilgilerini izleyebileceğiniz tek hata-ayıklama penceresidir. Gözlem Penceresini açmak için açılır menüden **View > Debug Windows > Watch Window**'u seçmelisiniz. Gözlem Penceresi PIC'in kayıtçılarını ve değişkenlerini, adresleri ve değerleri ile birlikte görüntüler. Değerler siz benzeştirme işlemine devam ettikçe değişecektir. Açılır menüyü kullanarak gözlemlemek istediğiniz parametreleri ekleyip çıkarabilirsiniz, son değiştirilen parametreler kırmızı ile gösterilecektir.



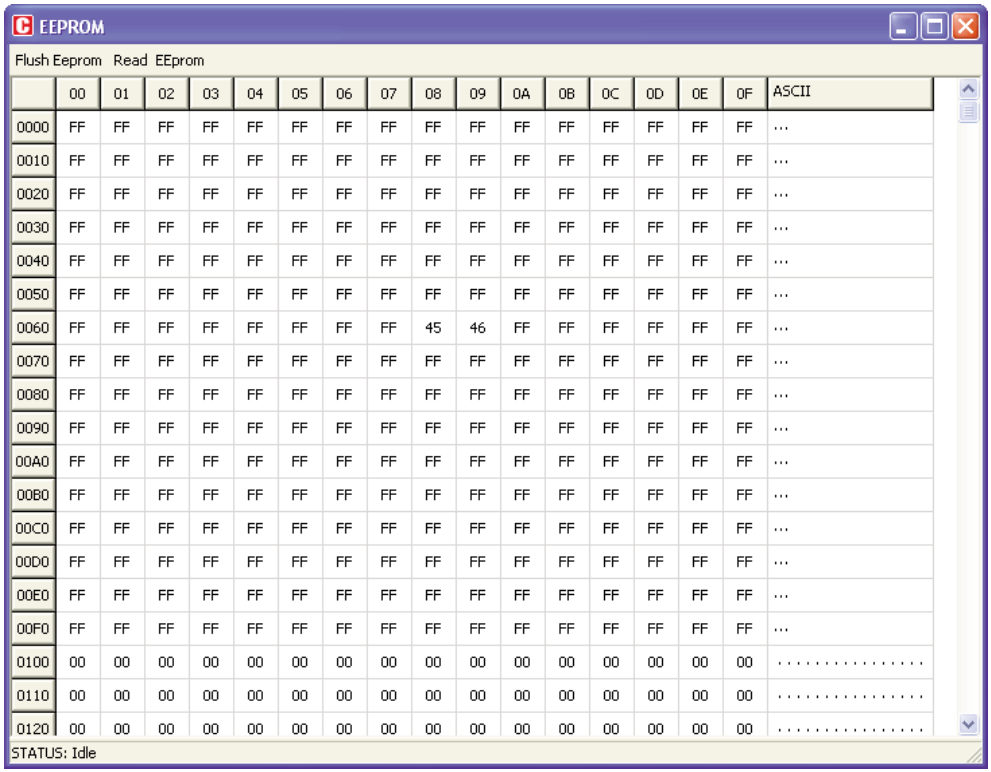
Herhangi bir parametreye çift tıkladığınızda Değer Düzenleme (Edit Value) penceresi açılacaktır. Bu pencerede istediğiniz değişken/yazmaca yeni değer verebilirsiniz. Ayrıca seçilen parametereyi ikilik, onaltılık, karakter veya ondalık düzende izlemek için seçim yapabilirsiniz.



EEPROM Penceresini Gözleme

mikro ICD EEPROM Penceresi açılır menüden , **View › Debug Windows › View EEPROM** yoluyla etkin hale gelir.

EEPROM Penceresi, PIC in iç EEPROM belleğine yazılan mevcut değerleri gösterir. EEPROM izleme penceresinin iki işlem butonu vardır- **Flush EEPROM** ve **Read EEPROM**. **Flush EEPROM** veriyi EEPROM penceresinden PIC'in iç EEPROM belleğine yazar. **Read EEPROM** veriyi PIC in iç EEPROM belleğinden okur ve daha sonra EEPROM penceresine yükler.

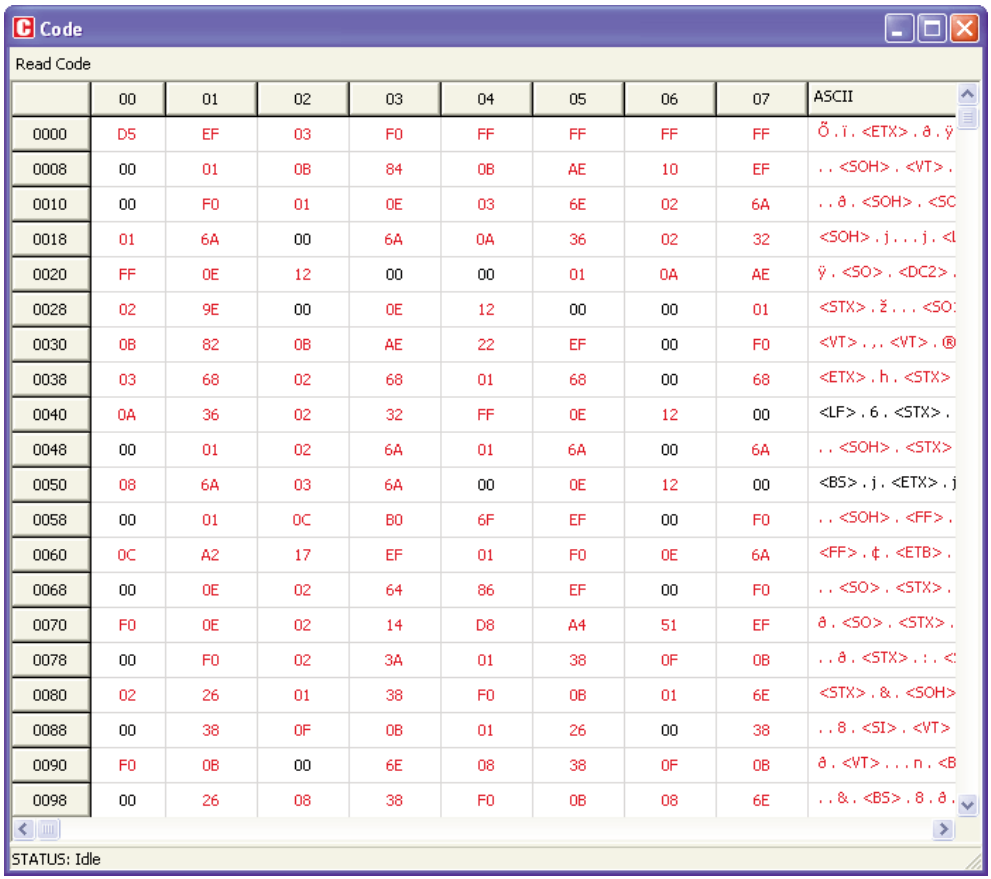


	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0060	FF	FF	FF	FF	FF	FF	FF	FF	45	46	FF	FF	FF	FF	FF	FF	...
0070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
00F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	...
0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Kod Gözlemleme Penceresi

mikro ICD Kod Gözlemleme Penceresi açılır menüden, **View › Debug Windows › View Code** yoluyla etkin hale gelir.

Kod Gözlemleme Penceresi PIC içerisine yazılan kodu (hex kod) gösterir. Kod Gözleme Penceresinin bir adet işlem butonu vardır.- **Read Code**. **Read Code** kodu PIC'ten okur daha sonrada Kod Gözleme Penceresine yükler.

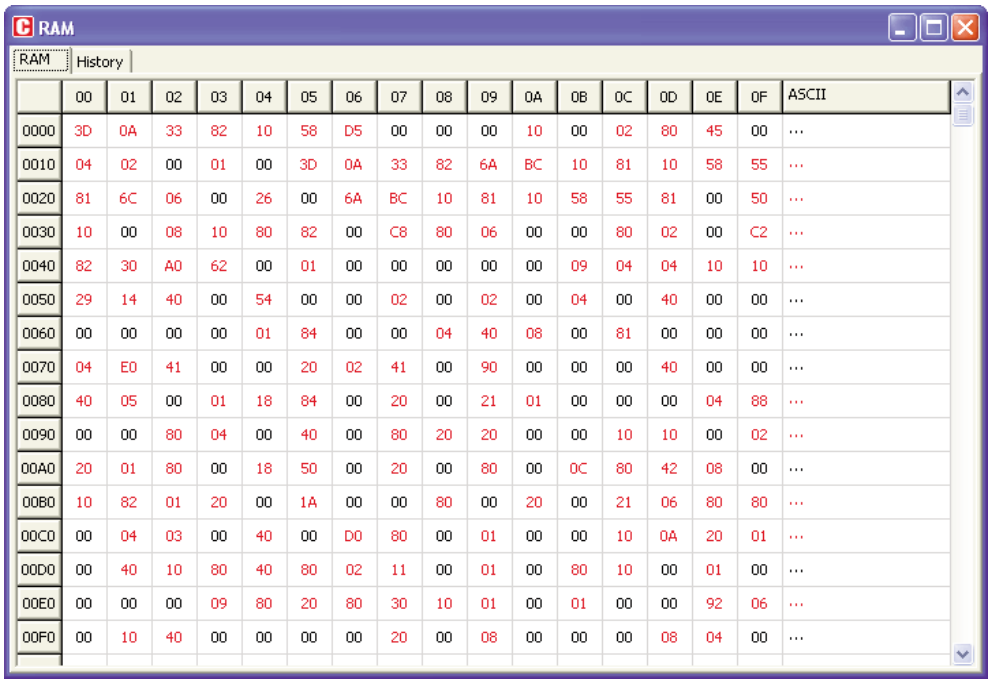


	00	01	02	03	04	05	06	07	ASCII
0000	D5	EF	03	F0	FF	FF	FF	FF	Õ . ï . <ETX> . ð . ÿ
0008	00	01	0B	84	0B	AE	10	EF	. . <SOH> . <VT> .
0010	00	F0	01	0E	03	6E	02	6A	. . ð . <SOH> . <SC
0018	01	6A	00	6A	0A	36	02	32	<SOH> . j . . j . <1
0020	FF	0E	12	00	00	01	0A	AE	ÿ . <SO> . <DC2> .
0028	02	9E	00	0E	12	00	00	01	<STX> . ž . . . <SO:
0030	0B	82	0B	AE	22	EF	00	F0	<VT> . , . <VT> . ®
0038	03	68	02	68	01	68	00	68	<ETX> . h . <STX>
0040	0A	36	02	32	FF	0E	12	00	<LF> . 6 . <STX> .
0048	00	01	02	6A	01	6A	00	6A	. . <SOH> . <STX>
0050	08	6A	03	6A	00	0E	12	00	<BS> . j . <ETX> . j
0058	00	01	0C	B0	6F	EF	00	F0	. . <SOH> . <FF> .
0060	0C	A2	17	EF	01	F0	0E	6A	<FF> . ¢ . <ETB> .
0068	00	0E	02	64	86	EF	00	F0	. . <SO> . <STX> .
0070	F0	0E	02	14	D8	A4	51	EF	ð . <SO> . <STX> .
0078	00	F0	02	3A	01	38	0F	0B	. . ð . <STX> . : . <:
0080	02	26	01	38	F0	0B	01	6E	<STX> . & . <SOH>
0088	00	38	0F	0B	01	26	00	38	. . 8 . <SI> . <VT>
0090	F0	0B	00	6E	08	38	0F	0B	ð . <VT> . . . n . <B
0098	00	26	08	38	F0	0B	08	6E	. . & . <BS> . 8 . ð .

RAM Penceresini Gözleme

RAM Penceresini açmak için açılır menüden **View > Debug Windows > View RAM'**i seçmelisiniz.

RAM Penceresi PIC'in RAM haritasını değerleri ile birlikte gösterir. Son değiştirilen parametreler **kırmızı** ile gösterilir.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
0000	3D	0A	33	82	10	58	D5	00	00	00	10	00	02	80	45	00	...
0010	04	02	00	01	00	3D	0A	33	82	6A	BC	10	81	10	58	55	...
0020	81	6C	06	00	26	00	6A	BC	10	81	10	58	55	81	00	50	...
0030	10	00	08	10	80	82	00	C8	80	06	00	00	80	02	00	C2	...
0040	82	30	A0	62	00	01	00	00	00	00	00	09	04	04	10	10	...
0050	29	14	40	00	54	00	00	02	00	02	00	04	00	40	00	00	...
0060	00	00	00	00	01	84	00	00	04	40	08	00	81	00	00	00	...
0070	04	E0	41	00	00	20	02	41	00	90	00	00	00	40	00	00	...
0080	40	05	00	01	18	84	00	20	00	21	01	00	00	00	04	88	...
0090	00	00	80	04	00	40	00	80	20	20	00	00	10	10	00	02	...
00A0	20	01	80	00	18	50	00	20	00	80	00	0C	80	42	08	00	...
00B0	10	82	01	20	00	1A	00	00	80	00	20	00	21	06	80	80	...
00C0	00	04	03	00	40	00	D0	80	00	01	00	00	10	0A	20	01	...
00D0	00	40	10	80	40	80	02	11	00	01	00	80	10	00	01	00	...
00E0	00	00	00	09	80	20	80	30	10	01	00	01	00	00	92	06	...
00F0	00	10	40	00	00	00	00	20	00	08	00	00	00	08	04	00	...

Sık yapılan hatalar

- mikro ICD aktif iken PIC'i programlamaya çalışmak.
- Release Debug tipi seçilmiş programda hata ayıklamaya çalışmak.
- Değişiklik yapılmış fakat derlenip PIC'e kaydedilmemiş programda hata ayıklamaya çalışmak.
- *Run to Cursor* **[F4]** ve *Toggle Breakpoints* **[F5]** fonksiyonlarını boş satırda çalıştırmaya çalışmak.

DİZGECİKLER (TOKENS)

Dizgecikler bir C programının derleyiciye anlam ifade eden en küçük parçalarıdır. Kod ayrıştırıcısı (Parser) giriş karakterlerini soldan sağa doğru tarayarak en uzun dizgeciği tespit eder.

mikroC şu tür dizgecikleri tanır:

- anahtar kelimeler (keywords)
- tanıtıcılar (identifiers)
- sabitler (constants)
- operatörler (operators)
- noktalama işaretleyicileri (genelde ayraçlar olarak bilinirler).

Dizgecik Ayrıştırma Örneği

Burada bir dizgecik ayrıştırma örneği mevcuttur. Aşağıdaki kod dizisini inceleyelim:

```
inter =  a+++b;
```

`inter` kelimesinin bir tek tanıtıcı olarak ayrıştırıldığına dikkat ediniz. Anahtar kelime `int` devamında ayrı `er` tanıtıcısı olarak ayrıştırılmamıştır.

Yukarıdaki kod şu şekilde de yazılmak istenebilir:

```
inter = a + (++b)
```

Ancak bu şekilde yazıldığında aynı sonucu vermez. Derleyici onu aşağıdaki gibi 6 dizgeciğe ayrıştıracaktır:

```
inter       // kimlikleyici
=           // atama operatörü
a           // kimlikleyici
++          // artım operatörü
+           // ek operatör
b           // kimlikleyici
;           // noktalı virgül ayracı
```

Bunun sebebi kod ayrıştırıcı mümkün olan en uzun dizgeciği bulmak ister ve bu yüzden '+++' ; '++' yı takip eden '+' şeklinde ayrışır.

SABİTLER (CONSTANTS)

Sabitler (constants) veya literaller (özdeğerler) belirli numerik veya karakter değerini gösteren dizgeciklerdir.

mikroC aşağıdaki seçenekleri destekler:

- tamsayı sabitleri,
- kayan nokta sabitleri,
- karakter sabitleri,
- karakter dizisi sabitleri (karakter dizisi özdeğerleri - literalleri),
- numaralama sabitleri.

Derleyici bir sabitin veri tipini kaynak kodun formatından veya numerik değer ipuçlarından ortaya çıkarır.

Tamsayı Sabitleri

Tamsayılar ikilik (binary-2'lik taban), onluk (decimal-10'luk taban) veya onaltılık (hexadecimal-16'lık taban) sayı sistemlerinde gösterilebilir. Herhangi bir son ek belirtilmediğinde, bir tamsayı sabitinin veri tipi kendi değerinden türetilir.

Long ve İşaretsiz (unsigned) Son-ekleri

L (veya l) son-eki bir sabiti `long` gibi gösterebilmek için sabitin sonuna eklenir. Benzer şekilde U (veya u) son-eki işaretsiz (`unsigned`) olacak sabitin sonuna eklenir. Gerekli ise hem L'yi hemde U'yu herhangi bir sırada ve formda aynı sabit için kullanabilirsiniz. : `ul`, `Lu`, `UL`, vs.

Herhangi bir son-ek'in yokluğunda, ilgili sabit aşağıdaki tiplerden değerinin sığdığı "en küçüğü" olarak kabul edilir : `short`, `unsigned short`, `int`, `unsigned int`, `long int`, `unsigned long int`.

Diğer taraftan:

Eğer sabit U veya u son-ek'ine sahipse, onun veri tipi aşağıdaki tiplerden sabitin değerinin sığdığı ilk tip olacaktır: `unsigned short`, `unsigned int`, `unsigned long int`.

Eğer sabit L veya l son-ek'ine sahipse, onun veri tipi aşağıdaki uygun tiplerden sabitin değerinin sığdığı ilk tip olacaktır: `long int`, `unsigned long int`.

Eğer sabit hem U hemde L son-eklerine sahipse, (`ul`, `lu`, `Ul`, `lU`, `uL`, `Lu`, `LU`, veya `UL`), veri tipi `unsigned long int` olacaktır.

Decimal (onluk-desimal) Sabitler

Desimal sabitler, -2147483648 ile 4294967295 arası kabul edilir. Sabit bu değeri geçerse "Out of Range" hatası verir. Desimal Sabitler sıfır karakteri ile başlamamalıdırlar. Sıfır karakteri ile başlayan bir sabit bir oktal (sekizlik) sabit olarak kabul edilir.

Herhangi bir son-ek in yokluğunda, veri tipi aşağıdaki örneklerde olduğu gibi kendi değerinden türetilir:

```
< -2147483648                   hata: Out of range!
-2147483648 .. -32769           long
-32768 .. -129                  int
-128 .. 127                     short
128 .. 255                      unsigned short
256 .. 32767                    int
32768 .. 65535                  unsigned int
65536 .. 2147483647             long
2147483648 .. 4294967295        unsigned long
> 4294967295                    hata: Out of range!
```

Hexadecimal (onaltılık) Sabitler

0x (veya 0X) ile başlayan tüm sabitler hexadecimal'dir. Herhangi bir son-ekin yokluğunda, hexadecimal sabitin veri tipi yukarıda gösterilen kurallara göre kendi değerinden türetilir. Mesela: 0xC367 bir işaretsiz tamsayı (`unsigned int`) şeklinde işlem görecektir.

Binary (ikilik) Sabitler

0b (veya 0B) ile başlayan tüm sabitler binary (ikilik) olarak kabul edilirler. Herhangi bir son-ekin yokluğunda, binary sabitin veri tipi yukarıda anlatılan kurallara göre kendi değerinden türetilir. Mesela: 0b11101 bir `short` şeklinde işlem görecektir.

Octal (sekizlik) Sabitler

0 ile başlayan tüm sabitler octal sabitlerdir. Eğer bir octal (sekizlik) sabit geçersiz olan 8 ve 9 değerlerini içerirse, hata bildirimi oluşturulur. Herhangi bir son-ekin yokluğunda, octal sabitin veri tipi yukarıda anlatılan kurallara göre kendi değerinden türetilir. Mesela: 0777 bir `int` olarak işlem görecektir.

Kayan Noktalı (Floating Point) Sabitleri

Bir kayan nokta sabiti şunlardan oluşur:

- Ondalık tamsayı,
- Ondalık nokta,
- Ondalık kısım,
- `e` veya `E` harfi ve 10'un üssü (bir işaretli tamsayı) (opsiyonel),
- Son-ek tipi: `f` veya `F` veya `l` veya `L` (opsiyonel).

Bir sayıda ondalık tamsayı kısmı veya ondalık kısım olmayabilir (Ancak ikisinden biri mutlaka olmalıdır.) Benzer şekilde bir sayı ya geleneksel yolla gösterilmeli (ondalık nokta ile) ya da bilimsel gösterim (yani `e veya E` harfi ve işaretli tamsayı olan üs kısmı) kullanılmalıdır.

Negatif kayan sabitler, eksi (-) ön-eki almış pozitif sabitler olarak kabul edilirler.

mikroC kayan noktalı sabitleri şu sınırlar arasında limitler:
`±1.17549435082E38 .. ±6.80564774407E38.`

mikroC kayan noktalı sabitler `double` tipindedirler. Unutmayın ki, mikroC'nin ANSI standardı uygulaması kayan (`float`) ve `double`'ı (ikisini birlikte `long double` değişken olarak) aynı tip olarak görür.

Karakter Sabitleri

Bir karakter sabiti, iki tekli tırnak işareti arasındaki bir veya daha fazla karakterden oluşur. Mesela `'A'`, `'+'`, veya `'\n'`. C'de, tek karakter sabitleri `int` veri tipine sahiptirler. Çoklu karakter sabitler, karakter dizisi sabitleri veya karakter dizisi literalleri olarak görülürler. Daha fazla bilgi için *Karakter Dizisi Sabitlerine* bakınız.

Kaçış Dizileri

Ters bölü (Backslash) karakteri (\), grafik olarak gösteril(e)meyen belli karakterlerin görsel gösterimlerine izin veren bir kaçış dizisini bildirmek için kullanılır. En çok kullanılan kaçış sabitlerinden biri yeni satır karakteridir (`\n`).

Bir ters bölü (Backslash), sekizlik veya onaltılık sayılarla bu değere karşılık gelen ASCII sembolü veya kontrol kodu göstermek için kullanılır. Örneğin, `'\x3F'` ü soru işareti için kullanırız. Herhangi bir kaçış dizisi içinde içinde üç adede kadar octal (sekizlik) rakam veya herhangi sayıda onaltılık rakamı kullanabilirsiniz, yeter ki ilgili sayısal değer `char` tipi için için geçerli aralıkta olsun (mikroC için 0 dan 0xFF'e kadar). Daha büyük sayılar derleyicinin şu hatayı vermesine neden olacaktır:"Numeric constant too large" ("Numerik sabit çok büyük").

Örneğin, `\777` sekizlik sayı maksimum izin verilen değer olan (`\377`) den çok büyüktür ve bu durumda bir hata görüntülenecektir. İlk karşılaşılan sekizlik veya onaltılık olmayan karakter bir sekizlik veya onaltılık kaçış dizisinde dizinin sonunu gösterir.
Not: ASCII ters bölü işareti için işletim sistemleri dizin yollarında olduğu gibi çift ters bölü (`\\`) kullanmalısınız.

Aşağıdaki tablo mikroC'de var olan kaçış dizilerini göstermektedir:

Dizi	Değeri	Karakter	Ne yapar
\a	0x07	BEL	Sesli zil
\b	0x08	BS	Geri al
\f	0x0C	FF	Form besleme
\n	0x0A	LF	Yeni satır (Sat. bes.)
\r	0x0D	CR	Taşıma Dönüşü
\t	0x09	HT	Tab (yatay)
\v	0x0B	VT	Dikey Tab
\\	0x5C	\	Ters bölü
\'	0x27	'	Tek Tırnak (kesme işareti)
\"	0x22	"	Çift tırnak
\?	0x3F	?	Soru işareti
\O		-	O = Karak. diz. sekizlik rakamlar
\xH		-	H =Karak. diz onaltılık rakamlar
\XH		-	H = Karak. diz onaltılık rakamlar

Karakter dizisi (String) Sabitleri (Dizgi sabitler)

Karakter dizisi sabitler, aynı zamanda karakter dizisi literalleri olarak da bilinirler. Bu sabitler, sabit karakter dizileri şeklinde depolanabilen özel bir tip sabittirler. Bir karakter dizisi literali çift tırnaklar ile sınırlandırılmış herhangi bir sayıdan oluşmuş karakter dizisidir:

```
"Bu bir karakter dizisidir."
```

null string veya boş string şu şeklilde yazılır: `""`. Bir literal karakter dizisi yani dizgi, verilen bir karakter dizisi artı bir son ”hiçlik“ karakteri olarak depolanır. Bir *null string* bir tek hiçlik karakteri olarak saklanır.

Çift tırnak arasındaki karakterler kaçış dizilerini de içerebilirler;

```
"\t\"Name\"\\\tAddress\n\n"
```

Arka arkaya ve aralarında sadece boşluk olan karakter dizileri parsing işlemi sırasında bitiştirilirler. Örnek olarak:

```
"Bu "   "guzel"
    " bir ornektir."
```

Şuna eşittir:

```
"Bu guzel bir ornektir."
```

Ters bölü (\) ile satır devamlılığı

Ters bölü (\ ) işaretini bir satırı aşan dizgilerde devam karakteri olarak kullanabilirsiniz:

```
"Bu bir tek  \
 satirlik karakter dizisi ornegidir."
```

Numaralama (Enumeration) Sabitleri

Numaralama sabitleri `enum` tip bildirimlerinde tanımlanan tanıtıcılardır. Tanıtıcılar kullanım kolaylığı için anımsatıcı kelimeler olarak seçilirler. Numaralama sabitleri `int` veri tipindedirler. Tamsayı sabitlerinin geçerli olduğu herhangi bir ifadede kullanılabilirler.

Örnek olarak:

```
enum haftanin_günleri { PAZ = 0, PZT, SALI, ÇAR, PER, CUM, CMT};
```

Tanıtıcılar (numaralamalar) `enum` bildiriminin kapsamı (scope) içinde tek olmalıdırlar. Negatif başlatıcılara izin verilir. Numaralandırmalar hakkında detaylı bilgi için `enum` bildirimlerine bakın.

İşaretçi (Pointer) Sabitleri

Bir işaretçi veya işaretlenen nesne `const` değiştiriciyle tanımlanabilir. `const` ile bildirilen herhangi bir şeyin değeri değiştirilemez. Sabit bir nesnenin atanamaz olma durumunu ihlal eden bir işaretçinin oluşturulmasına izin verilmez.

Sabit İfadeler

Bir sabit ifade, hesaplandığında sonucu bir sabit olan ve herhangi bir sabit (veya literal) veya sembolik sabit içeren ifadedir. Derleme zamanında değeri hesaplanır ve kendi veri tipi için gösterilen değerler aralığında bir sabit olarak değeri hesaplanmalıdır. Sabit ifadeler normal ifadeler gibi değerlendirilirler.

Sabit ifadeler yalnızca şunlardan oluşabilir: literaller, numaralama sabitleri (enumeration constants) , basit sabitler (sabit diziler veya yapılar yok), sizeof operatörleri.

Sabit ifadeler, operatörler bir `sizeof` operatörü içerisinde olmadığı müddetçe, aşağıdaki operatörlerden hiçbirini içermez: atama, virgül, azalma, fonksiyon çağırımı ve artma.

Sabit ifadeleri, sabitlerin geçerli olduğu her yerde kullanabilirsiniz.

ANAHTAR KELİMELER (KEYWORDS)

Anahtar kelimeler özel kullanım için ayrılmış kelimeler olup herhangi bir tanıtıcının (identifier) ismi olarak kullanılamazlar.

Standart C'in kullandığı anahtar kelimelere ek olarak tüm MCU SFR'leri de (Özel İşlevli Yazmaçlar) genel değişkenler olarak tanımlanmışlardır. Bu SFR'ler için özel olarak ayrılmış kelimeler kullanılır ve yeniden tanımlanamazlar. (Örneğin: TMR0, PCL, v.b). Özel kelimeleri editörde incelemek için Kod Asistanı kullanılabilir (CTRL+SPACE) veya "Tanımlanmış Değişken ve Sabitler" bölümüne bakabilirsiniz.

mikroC'de kullanılan anahtar kelimelerin alfabetik olarak listesi aşağıda verilmiştir:

```
asm             enum            signed
auto            extern          sizeof
break           float           static
case            for             struct
char            goto            switch
const           if              typedef
continue        int             union
default         long            unsigned
do              register        void
double          return          volatile
else            short           while
```

Ayrıca mikroC kendi kütüphanelerinde önceden tanımlanmış bir takım tanıtıcılar kullanmaktadır. Eğer kendinize kütüphane yaratmak isterseniz bunları kendi tanımlarınızla değiştirebilirsiniz. Daha fazla bilgi için mikroC Kütüphanelerine bakınız.

TANITICILAR (IDENTIFIERS)

Tanıtıcılar; fonksiyonlara, değişkenlere, sembolik sabitlere, kullanıcı tarafından tanımlanmış veri tiplerine ve etiketlere verilen özel isimler olup herhangi bir uzunlukta olabilirler. Tüm bu program elemanları yardım (help) konuları içinde nesne (object) olarak nitelendirilmişlerdir (Nesne'nin (object) anlamı object-oriented programlamadaki nesne (object) ile karıştırılmamalıdır).

Kimlikleyiciler, `a`'dan `z`'ye, `A`'dan `Z`'ye harfleri, altçizgi karakteri; '`_`' ve 0'dan 9'a kadar sayıları içererebilir. Tek sınırlama ilk karakterin bir harf veya altçizgi karakteri olması gereğidir.

Büyük-Küçük Harf Duyarlılığı (Case Sensivity)

mikroC tanıtıcıları büyük-küçük harflere duyarlı değildir; örneğin `Sum`, `sum`, ve `suM` aynı tanıtıcıyı belirler. Ancak, mikroC'nin gelecek versiyonları harf duyarlılığının aktif/pasif edilme seçeneğini sunacaktır. Şu andaki tek istisna ayrılmış (rezerve edilmiş) kelimeler olan `main` ve `interrupt`'ın küçük harfle yazılması zorunluluğudur.

Eşsizlik ve Kapsam

Tanıtıcı isimleri (belirtilen kurallara bağlı kalmak şartıyla) istenildiği gibi seçilebilirse de eğer aynı isim aynı kapsamdaki ve aynı isim havuzu birden fazla tanıtıcı için kullanılırsa hata oluşur.
Kapsam konusundan bağımsız olmak üzere ayrı isim havuzlarında bulunan tanıtıcılara aynı ismin verilmesinde sakınca yoktur. Kapsam konusunda daha fazla bilgi edinmek için "Kapsam (Scope) ve Görünürlük (Visibility)" bölümüne bakınız.

NOKTALAMA İŞARETLERİ (PUNCTUATORS)

mikroC noktalama işaretleri (ayrıca ayraçlar olarak da bilinirler) köşeli ayraç (yani "[" ve "]") , parantez (yani "(" ve ")") , küme ayracı (yani "{" ve "}") virgül, noktalı virgül, nokta, kesme işareti, eşittir işareti ve iki nokta üst üste karakterleridir. Çoğu noktalama işaretleyicileri aynı zamanda operatörler gibi işlev görürler.

Ayraçlar

Köşeli ayraçlar “[ ]” tek ve çok boyutlu dizilerin indekslerini gösterirler.

```
char ch, str[] = "mikro";

int mat[ 3][ 4] ;     /* 3 x 4 matrix */
ch = str[ 3] ;             /* 4'üncü eleman */
```

Parantezler

Parantezler “( )” ifadeleri gruplama, şartlı ifadeleri izole etme, fonksiyon dallanmaları ve fonksiyon parametreleri için kullanılır:

```
d = c * (a + b);   /* normal öncelikleri bastırır */
if (d == z) ++x;   /* şartlı ifadeler için gereklidir */
func();            /* Parametresiz fonksiyon çağırılması */
void func2(int n); /* Fonksiyon bildirimi, parametreleri */
```

Makroların koda eklenerek açılımları sırasında oluşabilecek karmaşaları önlemek amacıyla parantezlerin kullanılması tavsiye edilir.

```
#define CUBE(x) ((x)*(x)*(x))
```

Daha fazla bilgi için ‘İfadeler ve Operatör Önceliği’ bölümüne bakınız.

Küme Ayraçları (Braces)

Küme ayraçları{ } , birleşik bir deyimin (statement) başlama ve bitişini gösterir:

```
if (d == z) {
  ++x;
  func();
}
```

Kapatma küme ayracı birleşik deyim için bir sonlandırıcı olarak hizmet görür. Bu nedenle küme ayracından (yani }) sonra bir noktalı virgül kullanılmasına gerek yoktur (Bir yapı (structure) bildirimi sonu hariç).

```
if (deyim)
   { ... };     /* gecersiz noktali virgul! */
else
   { ... };
```

Daha fazla bilgi için “Birleşik Deyimler” kısmına bakınız.

Virgül (Comma)

Virgül (,) bir fonksiyonun bağımsız değişken listesinin elemanlarını ayırmak için kullanılır:

```
void func(int n, float f, char ch);
```

Aynı zamanda virgül, virgül ifadelerinde bir operatör olarak kullanılır. Her iki virgülün bir birine karışması sorun yaratmaz, fakat birbirinden ayırt etmek için parantez kullanmalısınız. Unutmayın ki (exp1, exp2) parantezindeki ifadelerin ikisi de hesaplanır fakat değerlendirme sonucu ikincisine eşittir.

```
/* iki bagimsiz degiskenle func fonksiyonunu cagirir */
func(i, j);

/* func fonksiyonunun iki farkli bagimsiz degisken ile
çagrilmasina farkli bir ornek! */
func((exp1, exp2), (exp3, exp4, exp5));
```

Noktalı Virgül (Semicolon)

Noktalı virgül (;) bir deyim sonlandırıcısıdır. Noktalı virgül ile devam eden her geçerli C ifadesi (boş ifadeler dahil) bir deyim gibi yorumlanır ve ifade deyimi olarak bilinirler. İfade hesaplanır ve değeri ihmal edilir. Eğer ifade deyiminin yan etkileri yoksa, mikroC onu ihmal edebilir.

```
a + b;      /*  a + b seklinde deger kazanir, fakat deger
            ihmal edilir (atilir) */
++a;        /* a uzerinde bir yan etki, fakat ++a nin degeri
            ihmal edilir (atilir) */
;           /* bos ifade veya hukumsuz deyim */
```

Noktalı virgül bazen bir boş deyim oluşturmak için kullanılır.

```
for (i = 0; i < n; i++) ;
```

Daha fazla bilgi için Deyimlere (Statements) bakınız.

İki nokta üst-üste (Colon)

İki nokta üst-üste (:) bir etiketli deyimi göstermek için kullanılır. Örnek:

```
start:  x = 0;
  ...
goto start;
```

Etiketler, “Etiketli Deyimler” bölümünde ele alınmıştır.

Yıldız İşareti (Asterisk) ve İşaretçi bildirimi (Pointer Declaration)

Bir bildirim (declaration) içerisindeki yıldız işareti (*) bir tipe bir işaretçi oluşturulmasını sağlar.

```
char *char_ptr;  /* bir karaktere isaretci tanimlama */
```

Aynı zamanda yıldız işaretini, bir işaretçinin işaret ettiği yerdeki değere ulaşmak için ya da çarpma operatörü olarak kullanabilirsiniz:

```
i = *char_ptr;
```

Daha fazla bilgi için “İşaretçiler” bölümüne bakınız.

Eşittir işareti

Eşittir işareti (=) değişken tanımları ile ilk değer listesi arasına koyulur:

```
int test[5] = {1, 2, 3, 4, 5};
int x = 5;
```

Eşittir işareti aynı zamanda ifadelerde atama operatörü olarak kullanılır:

```
int a, b, c;
a = b + c;
```

Daha fazla bilgi için “Atama Operatörleri” ne bakınız.

Diyez işareti (Pound Sign- #)

Diyez işareti, bir satırda beyaz boşluk olmayan ilk karakter olarak geldiğinde bir ön-işlemci direktifini belirtir. Bir derleyici işlemi anlamına gelir, kod üretimiyle ilgili olması zorunlu değildir. Daha fazla bilgi için “Ön-işlemci Direktifleri” bölümüne bakınız.

`#` ve `##` aynı zamanda ön-işlemci tarama aşamasında, dizgeciklerin bir başkasıyla değiştirilmesi ve birleştirilmesinde operatör olarak kullanılırlar. “Ön-işlemci Operatörleri”ne bakınız.

NESNELER VE SOL-DEĞERLER (LVALUES)

Nesneler

Bir nesne, belleğin sabit veya değişken bir değeri (veya değerler grubunu) tutabilen belli bir bölgesidir. Karmaşayı önlemek gerekirse, bu kullanılan nesne kelimesi nesneye yönelik (object-oriented) dillerdeki daha genel kullanılan terimden farklıdır. Bizim tanımladığımız kelime; fonksiyonları, değişkenleri, sembolik sabitleri, kullanıcı tanımlı veri tiplerini ve etiketleri içerir.

Her değer ilgili bir isim ve veri tipine sahiptir. İsim nesneye erişim için kullanılır. Bu isim basit bir tanıtıcı veya tek olarak nesneye referans olan karmaşık bir ifade olabilir.

Nesneler ve Bildirimler

Bildirimler, tanıtıcılar (identifier) ve nesneler arasında gerekli köprüyü kurarlar. Her bildirim, bir tanıtıcıyı bir veri tipi ile ilişkilendirir.

Tanıtıcıları nesnelerle ilişkilendirmek, her tanıtıcının iki özniteliğe sahip olmasını gerektirir: depolama sınıfı ve tip (veri tipi). mikroC derleyicisi bu öz nitelikleri kaynak kod içerisinde kendinden belli veya açıkça bildirimlerden ortaya çıkarır. Genelde, sadece tip açık olarak belirtilir ve depolama sınıfı belirteci (specifier) otomatik olarak oto değer varsayılır.

Genelden söz etmek gerekirse, bir tanıtıcı bir program içerisinde, kaynak kodu içerisindeki kendi bildirim noktasından daha önce geçerli bir şekilde kullanılamaz. Bu kurala geçerli istisnai durumlar; etiketler (labels), bildirilmemiş fonksiyonların çağrılmaları ve yapı (struct) veya birlik (union) etiketleridir.

Tanımlanabilen nesnelerin erimleri şunları içerir:

değişkenler; fonksiyonlar; tipler; diğer tiplerin dizileri; yapılar; birlikler ve numaralama (enumeration) etiketleri; yapı (structure) üyeleri; birlik (union) üyeleri; numaralama (enumeration) sabitleri; deyim etiketleri (labels); ön-işlemci makroları.

Bildirim sözdiziminin özyineli yapısı karmaşık bildiricilere izin verir. Muhtemelen karmaşık nesneleri biçimlendirirken okunaklılığı artırmak için `typedefs` kullanmak isteyeceksiniz.

Sol-değerler (Lvalues) (Nesne gösteren ifadeler)

Bir Ldeğeri (L=left'in (sol) kısaltması) nesne belirleyicisidir: bir nesneyi belirten bir ifadedir. Ldeğer ifadesine bir örnek *P ifadesidir, burada *P boş olamayan bir işaretçinin değerine ulaşan herhangi bir ifadedir. Değiştirilebilir bir ldeğer "bellek içerisindeki geçerli bir şekilde değiştirilebilen ve erişimi sağlanabilen bir nesne" ile ilişkili "bir ifade veya bir tanımlayıcıdır". Örneğin bir sabiti gösteren sabit bir işaretçi değiştirilebilir bir ldeğeri değildir. İşatetçiler için ek bilgi verirsek; bir sabiti gösteren alelade bir işaretçi değiştirilebilir (fakat onun gösterdiği değer değiştirilemez).

Eskiden, "l" "sol"un (left'in) kısaltması idi ve bu şu anlama gelmekteydi: Kural olarak bir ldeğer bir atama deyiminin solunda (yani atamayı alan kısımda) var olabilirdi. Ama günümüzde sadece değiştirilebilir bir ldeğer kurallara göre bir atama deyiminin solunda yer alabilir. Örnek olarak: eğer a ve b belleğe düzgün olarak yerleştirilmiş ve sabit olmayan tamsayı tanıtıcıları ise, ikiside değiştirilebilir ldeğerleri'dirler ve a =1 ve b=a+b şeklinde atamaları geçerlidir. .

Sağ-değerler (Rvalues)

a+b ifadesi bir ldeğeri değildir: a+b=a geçersizdir çünkü soldaki ifade bir nesne ile ilişkili değildir. Bu tür ifadelere rdeğerleri denir (r = right'ın (sağ) kısaltması).

KAPSAM VE GÖRÜNÜRLÜK (SCOPE AND VISIBILITY)

Kapsam (Scope)

Tanıtıcının kapsamı, tanıtıcıyı kullanarak nesnesine erişebildiğimiz program kısmıdır. Çeşitli kapsam kategorileri mevcuttur: blok (veya yerel), fonksiyon, fonksiyon prototipi ve dosya. Bunlar tanıtıcının nerede ve nasıl bildirildiği ile ilişkilidir.

Blok Kapsamı (Block Scope)
Blok kapsamlı bir tanıtıcının kapsamı bildirim noktasında başlar ve bildirimi içeren bloğun sonunda sona erer (bu tür bloklara kapsama blokları denir). Bir fonksiyon tanımıyla kullanılan parametre bildirimleri de fonksiyon gövdesinin kapsamı ile sınırlı, blok kapsamına sahiptirler.

Dosya kapsamı (File Scope)
Dosya kapsamı tanıtıcıları aynı zamanda "global" olarak da bilinirler ve dosyadaki tüm blokların dışında tanımlanırlar. Kapsamları bildirim noktasından kaynak dosyasının sonuna kadardır.

Fonksiyon Kapsamı
Fonksiyon kapsamına sahip tanıtıcılar sadece deyim etiketleridirler (labels). Etiket isimleri etiketin bildirildiği fonksiyonun içinde herhangi bir yerde *goto* deyimleri ile kullanılabilir. *label_name* ve devamında bir kod deyimi ile etiketler kolayca tanımlanır. Etiket isimleri bir fonksiyon içerisinde bildirim yapılırken tek ve eşsiz olmalıdır.

Fonksiyon Prototipi Kapsamı
Bir fonksiyon prototipindeki parametre bildirim listesindeki bildirilen tanıtıcılar fonksiyon prototipi kapsamına sahiptirler. Dikkat ediniz fonksiyon tanımının parçasıdırlar demiyoruz. Bu kapsam fonksiyon prototipinin sonunda sonlanır.

Görünürlük (Visibility)

Bir tanıtıcının (identifier) görünürlüğü, tanıtıcının ilgili nesnesine (kurallı olarak) ulaşılabilen kaynak kod programı bölgesidir.

Genelde kapsam ve görünürlük üstüste kesişir. Ancak öyle durumlar olabilir ki bir nesne çift bildirim (yani aynı isimli diğer bir tanıtıcı) yüzünden geçici olarak erişilemez (yani saklı) kalabilir. Bu durumda nesne hala vardır. Ancak diğer isimdaş tanıtıcının kapsamı sona erene kadar orijinal tanıtıcı ile ilgili nesneye erişilemez.

Teknik olarak görünürlük kapsamı aşamaz, ancak kapsam görünürlüğü aşabilir (yani kapsayabilir). Aşağıdaki örneğe bakınız:

```
void f (int i) {
 int j;            // otomatik değişken
 j = 3;            // int i ve j kapsam ve gorunurluk icinde

   {               // ic-ice gecmis blok
    double j;      // j ic blok icerisinde yerel isim
    j = 0.1;       // i ve double tipindeki j gorulebilir;
                   // int j = 3 kapsam icinde fakat gizli
   }
                   // artik double tipindeki j kapsam disinda
 j += 1;           // int j gorulebilir ve degeri = 4'tur
}
// i ve j, her ikisi de kapsamin disinda
```

İSİM UZAYLARI (NAME SPACES)

İsim uzayı, içinde bir tanıtıcının adaşsız olması gereken alandır. Bu isim uzayında aynı isimli başka tanıtıcı olamaz. C, tanıtıcıların dört farklı kategorisini kullanır:

Goto etiket isimleri
Tanımlandıkları fonksiyon içerisinde aynı isimli başka etiket olmamalıdır.

Yapı (Structure), birlik (union) ve numaralama (enumeration) isimleri
Tanımlandıkları bloklarda aynı isimli bildirimler olmamalıdır. Fonksiyonların dışında bildirimi yapılanlar için de aynı isimli başka bildirim olmamalıdır.

Yapı ve birlik üyelerinin isimleri
Tanımlandıkları yapı ve birlik içerisinde aynı isimli bildirimler olmamalıdır. Farklı yapılardaki adaş üye isimleri için herhangi bir tip veya kayma (offset) kısıtlaması yoktur.

Değişkenler, tip tanımları (typedefs), fonksiyonlar ve numaralama üyeleri
Bunlar tanımlandıkları kapsam içerisinde tek olmalıdırlar. Harici bildirimi yapılan tanıtıcılar, harici tanımlı değişkenler arasında tek olmalıdırlar.

Kapsam kurallarından bağımsız olarak, farklı isim uzayları için isimlerin adaş olması kurallara uygundur.

Örnek:

```
int mavi = 73;

{  // bir blok ac
    enum renkler {  siyah, kırmızı, yeşil, mavi, beyaz }  c;
    /* Enum mavi disaridaki int mavi bildirimini baskilar*/

    struct renkler {  int i, j; } ;
    // GECERSIZ: renkler yapisinin adas ismi (enum) var

    double kirmizi = 2;
    // GECERSIZ: kirmizinin tekrar tanimlanmasi
}

mavi = 37;  // Tekrar int mavi kapsamina donus
```

SÜRE (DURATION)

Süre, depolama sınıfı ile yakından ilgilidir. Bildirimi yapılan tanıtıcıların sahip oldukları gerçek fiziksel nesnelerin bellek içerisinde var oldukları süreyi tanımlar. Öte yandan biz derleme zamanı ve çalışma zamanı nesnelerini de birbirinden ayırırız. Değişkenler, örnek olarak, typedefler ve tiplerin aksine, çalışma süresince gerçek bellek yerleştirmesine sahiptirler. İki çeşit süre vardır: *statik* ve *yerel (local)*.

Statik Süre

Program çalışması başlar başlamaz statik süreli nesnelere bellek ayırılır. Bu depolama tahsisi program sonlanıncaya kadar sürer. Statik süre nesneleri genellikle (yürürlükteki bellek modeline bağlı olarak tahsis edilen) sabit ve belli veri parçalarında tutulurlar. Tüm globaller statik süreye sahiptirler. Tüm fonksiyonlar, her nerede tanımlanırlarsa tanımlansınlar, statik süreli nesnelerdir. Diğer değişkenlerin statik süreli olmaları ise `static` veya `extern` depolama sınıf belirteçlerini kullanarak sağlanabilir.

mikroC'de, statik süre nesnelerine, özellikle verilen bir ilk değer yoksa, sıfır olarak ilk değer verilmez.

Bir nesne statik süreye ve aynı zamanda yerel kapsama sahip olabilir. Bir sonraki sayfadaki örneğe bakınız.

Yerel Süre

Yerel süre nesneleri aynı zamanda otomatik nesneler olarak da bilinirler. Yığıtlarda (stack) veya yazmaçlarda (register), çevreleyen bloğa veya fonksiyona girildiğinde oluşturulurlar. Program bloktan çıktığında ve fonksiyonu terkettiğinde onlar da bellekten serbest bırakılırlar. Yerel süre nesnelerine özellikle ilk değer verilmelidir; aksi taktirde, içerikleri tahmin edilemez.

`auto` depo sınıf belirleyicisi, yerel süre değişkenlerinin tanımları yapıldığında kullanılabilir, fakat genellikle gereksizdir. Çünkü, bir blok içerisinde yapılan değişken tanımları için zaten `auto` varsayılmaktadır.

Yerel süreli bir nesne aynı zamanda yerel kapsama da sahiptir. Çünkü, kendini kapsayan bloğun dışında var olamaz ki. Bunun tersi doğru değildir: bir yerel kapsam nesnesi statik süreye sahip olabilir.

Aşağıda yerel kapsamlı, fakat farklı süreli iki nesnenin örneği yer almaktadır:

```
void f() {
  /* yerel sureli "var"; f her cagrildiginda a'ya ilk deger ver */
  int a = 1;

  /* statik sureli "var"; sadece f ilk cagrildiginda b'ye ilk değer
ver */
  static int b = 1;

  /* kontrol noktasi! */
  a++;
  b++;
}

void main() {
  /* Kontrol noktasinda, sunlara sahip olacagiz : */
  f(); // a=1, b=1, ilk aramadan sonra,
  f(); // a=1, b=2, ikinci aramadan sonra,
  f(); // a=1, b=3, ucuncu aramadan sonra,
       // v.b.
}
```

TİPLER (TYPES)

C tam anlamıyla tiplendirilmiş bir dildir. Yani her nesne, her fonksiyon ve her ifadenin tipleri derleme anında kesin olarak belirlenmiştir. Unutmayın ki, C yalnızca numerik tiplerle çalışır.

Tipler şu işlere yararlar:

- en başta gerekli bellek yerleşiminin doğru hesaplanmasını sağlama,
- sonradan yapılan erişimler sırasında nesne içerisinde bulunan bit örneklerini doğru yorumlama,
- çoğu tip-kontrolü durumunda, geçersiz atama işlemlerini tesbit etme.

mikroC, değişik boyutlarda işaretli ve işaretsiz tamsayılar, değişik duyarlılıkta kayan noktalı sayılar, diziler, yapılar, ve birlikler dahil olmak üzere standart (önceden tanımlı) veya kullanıcı tarafından belirtilmiş bir çok veri tipini destekler. Ek olarak, bellekte bu nesnelerin çoğunun işaretçileri kurulabilir ve işlenebilir.

Tipler bir nesne için ne kadar bellek ayrılacağını belirler. Ayrıca nesnenin depolama bölgesinde bulunan bit örneklerinin program tarafından nasıl yorumlanacağını da belirler. Verilen bir veri tipi, bir değerler grubu olarak görülebilir (daha çok uygulamaya bağlı). Bir veri tipi (genellikle uygulamaya bağlı olarak) bu tip için geçerli bir değerler kümesi olarak görülebilir. Ayrıca, bu değerler üzerinde geçerli işlemler seti de bu kümeye dahildir. Not olarak bir derleme-zamanı işlemi olan `sizeof` işlemi, herhangi bir standart veya kullanıcı tanımlı tipin boyutunu byte olarak belirlemenizi sağlar.

mikroC standart kütüphaneleri ve sizin kendi program ve başlık dosyalarınız, ne olduğu tartışmasız olarak belli tanıtıcılar (veya onlardan türetilmiş ifadeler) ve tipler sağlamalıdır. Böylelikle, mikroC, programınız içerisinde her aktif nesneye karşılık gelen bellekteki bit örneklerine erişebilir, onları yorumlayabilir ve (eğer mümkünse) onları değiştirebilir.

Tip Kategorileri

Temel (*fundamental*) tipler, daha küçük parçalara ayrılamayan tipleri ifade ederler. Bazen yapılandırılmamış tipler olarak da adlandırılırlar. Temel tiplerin bazıları `void`, `char`, `int`, `float`, ve `double`'dır ve bunların bazılarına `short`, `long`, `signed`, ve `unsigned` ek olarak gelebilir. Öte yandan türetilmiş (*derived)* tipler, aynı zamanda yapılandırılmış tipler olarak da bilinirler. Türetilmiş tipler; diğer tiplerin işaretçileri, diğer tiplerin dizileri, fonksiyon tipleri, yapılar ve birliklerdir.

TEMEL TİPLER (FUNDAMENTAL TYPES)

Aritmetik Tipler

Aritmetik tip veri tipleri aşağıdaki anahtar kelimelerden oluşturulur: `void`, `char`, `int`, `float`, ve `double`, ayrıca bunlara eklenen `short`, `long`, `signed`, ve `unsigned` ön-ekleri ile. Bu anahtar kelimelerden yararlanarak tümleşik ve kayan noktalı tipler oluşturabilirsiniz. Tipler hakkında bilgi bir sonraki sayfada verilmiştir.

Tümleşik Tipler (Integral Types)

`char` ve `int` tipleri, çeşitli türevleri ile birlikte tümleşik veri tipini oluştururlar. Değişik türevleri `short`, `long`, `signed`, ve `unsigned` ön-eklerinden birinin kullanılması ile oluşturulur.

Bir sonraki sayfadaki tabloda tümleşik tipler gösterilmiştir. Parantez içerisindeki anahtar kelimeler orada kullanılmasalar da olur ve genelde de kullanılmazlar.

`signed` ve `unsigned` değiştiricileri hem `char`'a hemde `int`'e eklenebilir. unsigned ön-ekinin olmadığı durumlarda, tümleşik tipler için otomatik olarak `signed` varsayılır. Tek istisna, `char` normalde `unsigned` olarak varsayılır. `signed` ve `unsigned`, anahtar kelimeleri, tek başlarına kullanılabilirler ve kullanıldıklarında, `signed int` veya `unsigned int` anlamına gelirler.

`short` ve `long` değiştiricileri sadece `int`'e uygulanabilir. `short` ve `long` anahtar kelimeleri kendi başlarına kullanıldıklarında `short int` ve `long int` anlamına gelirler.

Kayan noktalı Tipler

`float` ve `double` tipleri, `long double` türevleri ile birlikte, kayan-noktalı tipleri oluştururlar. mikroC'nin ANSI standardı uygulamaları her üç tipi de aynı gibi hesaba katar.

mikroC içerisinde kayan-nokta Microchip'in AN575 32-bitlik formatının kullanılması ile uygulanmıştır. (IEEE 754 uyumlu).

Aritmetik Tipler:

Tip	Boyut	Aralık
`(unsigned) char`	8-bit	0 .. 255
`signed char`	8-bit	- 128 .. 127
`(signed) short (int)`	8-bit	- 128 .. 127
`unsigned short (int)`	8-bit	0 .. 255
`(signed) int`	16-bit	-32768 .. 32767
`unsigned (int)`	16-bit	0 .. 65535
`(signed) long (int)`	32-bit	-2147483648 .. 2147483647
`unsigned long (int)`	32-bit	0 .. 4294967295
`float (kayan)`	32-bit	±1.17549435082E-38 .. ±6.80564774407E38
`double`	32-bit	±1.17549435082E-38 .. ±6.80564774407E38
`long double`	32-bit	±1.17549435082E-38 .. ±6.80564774407E38

Numaralamalar (Enumerations)

Bir numaralama veri tipi, uygun sembolik isimlerle bir soyut, ayrı değerler setinin gösterilmesi için kullanılır.

Numaralama Tanımı

Numaralamalar şu şekilde bildirilirler:

```
enum isim { numeral_listesi} ;
```

Burada, "isim" numaralamanın opsiyonel ismidir; *enumeration-list* her biri ayrı değer olan numaralama elemanları listesidir. Parantezler içerisinde listelenen numaralandırıcılar aynı zamanda numaralama sabitleri olarak bilinirler. Her birine belirlenmiş sabit bir tümleşik değer atanır. Özel olarak verilen ilk değerlerin yokluğunda, ilk numaralandırıcı sıfıra ayarlanır. Takip eden her numaralandırıcı bir öncekinden bir fazla değere ayarlanır.

`enum` tipi değişkenler diğer tiplerdeki değişkenler gibi tanımlanırlar. Örnek olarak:

```
enum renkler { siyah, kırmızı, yeşil, mavi, mor, beyaz}  c;
```

tek bir tümleşik tip kurar, `renkler`, bu tipte bir değişken olan `c`, ve sabit tamsayı değerleri ile birlikte bir takım numaralandırıcılar (`siyah = 0`, `kırmızı= 1`, ...). C'de, numaralama değişkenine herhangi bir `int` tip atanabilir ve başka bir "tip kontrolü" yapılmaz. Örneğin:

```
r = red;     // GECERLI
r = 1;       // bu da GECERLI, ayni anlama gelir
```

Açıkça belirtilen ilk değerler ile bir veya daha fazla numaralandırıcıya özel değerler atayabilirsiniz. Kullanılan ilk değer ifadesi pozitif veya negatif tamsayı sonucunu veren (tabi tamsayı çevrim kuralları uygulandıktan sonra) herhangi bir ifade olabilir. Bu numaralandırıcıdan sonraki özel olarak ilk değeri verilmemiş numaralandırıcılar artı bir fazla değere sahip olacaklardır. Bu değerler normalde tek ve eşsizdir, fakat çiftlere de izin verilir.

Sabitlerin sıraları özel bir şekilde tekrar düzenlenebilir. Örnek olarak:

```
enum renkler {  siyah,      // deger 0
                kirmizi,    // deger 1
                yesil,      // deger 2
                mavi=6,     // deger 6
                mor,        // deger 7
                beyaz=4 };  // deger 4
```

İlk değer verme ifadesi önceden bildirilmiş numaralandırıcıları da içerebilir. Örnek olarak aşağıdaki bildirim:

```
enum bellek_boyutları {  bit = 1, yarim_byte = 4 * bit,
              byte = 2 *  yarim_byte, kilobyte = 1024 * byte };
```

`yarim_byte` 4 değerini elde edecek, `byte` 8 değerini ve `kilobyte` 8192 değerini elde edecektir.

Anonim Enum Tipi

Önceki bildirimimizde, `renkler` tanıtıcısı istenirse sonradan kullanılacak olan `renkler` tipindeki başka numaralama değişkenlerinin tip bildirimi için de kullanılabilecek seçimlik bir isimdi.

```
enum renkler bg, sinir;  //  bg ve sinir degiskenlerini bildirir
```

Yapı ve birliklerin tanımlarında olduğu gibi, eğer bu enum tipinden başka gerekli değişken yoksa, isim (tag) koymayabilirsiniz:

```
/* Anonim enum tipi: */
enum { siyah, kirmizi, yesil, mavi, mor, beyaz}  renk;
```

Numaralama Kapsamı

Numaralama isimleri (etiketleri) yapı ve birlik etiketleri ile aynı isim uzayı gurubuna girerler. Numaralayıcılar, ise alelade değişkenler ile aynı isim uzayındadırlar. Daha fazla bilgi için “İsim Uzayları” bölümüne bakınız.

Void Tip

`void`, herhangi bir değerin olmadığını gösteren özel bir tip gösterme şeklidir. `void`'in nesnesi yoktur, buna karşılık olarak, daha karmaşık tipler türetmek için kullanılır.

Void Fonksiyonları

Eğer fonksiyon bir değer döndürmüyor ise, void anahtar kelimesi fonksiyonun geri-dönüş tipi olarak kullanılır. Örnek olarak:

```
void print_temp(char temp) {
  Lcd_Out_Cp("Sicaklik:");
  Lcd_Out_Cp(temp);
  Lcd_Chr_Cp(223);  // derece karakteri
  Lcd_Chr_Cp('C');
}
```

Eğer fonksiyon herhangi bir parametre almıyorsa, `void` bir fonksiyonun başlığında kullanılır. Alternatif olarak, sadece boş parantezler de yazabilirsiniz:

```
void main(void) {  //  veya sadece void main()
  ...
}
```

Soysal (Generic) İşaretçiler

İşaretleyiciler `void` olarak tanımlanabilirler ve bu işaretçiler herhangi bir tipi işaret edebilirler. Bu işaretçilere *soysal*'da denir.

TÜRETİLMİŞ TİPLER (DERIVED TYPES)

Türetilmiş tipler aynı zamanda yapısal tipler olarak da bilinirler. Bu tipler daha karmaşık kullanıcı tanımlı tiplerin oluşturulmasında yapı taşları olarak kullanılırlar.

Diziler (Arrays)

Diziler, en basit ve en çok kullanılan yapısal tiplerdir. Dizi tipindeki değişken, genellikle aynı tipteki nesnelerin bir dizisidir. Bu nesneler, bir dizinin elemanlarını temsil ederler ve dizi içerisindeki pozisyonlarına göre tanımlanırlar. Bir dizi, tüm elemanlarını tam tamına alacak genişlikte, bitişik bellek bölgelerinden oluşur.

Dizi Bildirimi

Dizi bildirimi değişken bildirimine benzer, ek olarak tanıtıcıdan sonra parantezler eklenir:

tip dizi_ismi[*sabit-ifade*]

Yukarıdaki ifade ile *tip* elemanlarından oluşan *dizi_ismi* adındaki bir dizinin bildirimi yapılmış olur. *tip* skaler bir tip, kullanıcı tanımlı bir tip, işaretçi, numaralama veya başka bir dizi olabilir. Parantezler içerisindeki *sabit-ifade* nin sonucu dizi içerisindeki elemanların sayısını belirtir. Eğer dizi bildiriminde (köşeli parantezlerin içerisinde) bir ifade verilmişse, bu bir pozitif sabit tamsayı olarak değer kazanmalıdır. Bu değer dizi içerisindeki eleman sayısıdır.

Bir dizinin her elemanı, sıfırdan toplam eleman sayısının bir eksiğine kadar numaralandırılır. Eğer eleman sayısı ‘n’ ise, dizinin elemanları şu şekilde numaralar alır.
tip cinsinden ve dizi_ismi[0] ‘dan *dizi_ismi*[n-1] ‘e kadar.

Aşağıda bir kaç dizi tanımlamaları yer almaktadır. :

```
#define MAX = 50

int vector_one[ 10];      /* 10 tamsayili bir dizi */
float vector_two[ MAX];   /* 50 kayan noktalı (float) dizi */
float vector_three[ MAX - 20]; /* 30 kayan noktali bir dizi
*/
```

Dizilere İlk Değer Vermek

Dizilere, bildirim sırasında, kaşlı parantezler içerisinde virgüllerle birbirlerinden ayrılan değerler dizisi kullanarak ilk değerleri verilebilir. Bir diziye bildirim sırasında ilk değerler atanacak ise eleman sayısını atlayabilirsiniz - Dizinin büyüklüğü otomatik olarak atanan eleman sayısına göre ayarlanacaktır. Örnek olarak:

```
/* her ay icerisindeki gunleri tutan bir dizi: */
int gunler[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

/* Bu tanimlama bir onceki ile aynidir */
int gunler[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Eğer hem dizi uzunluğunu ve ilk değerleri belirtirseniz, ilk değerlerin sayısı belirtilen dizi uzunluğunu geçmemelidir. Tam tersi ise geçerlidir, ancak ilk değer verilmemiş elemanlara bellekte çalışma zamanında (run time) hangi değerler varsa o değerler atanacaktır.

`char` dizileri olduğunda, daha kısa bir karakter dizisi literali biçimi kullanabilirsiniz. Örneğin:

```
/* Her iki tanimlama da ozdestir: */
const char msg1[] = {'T', 'e', 's', 't', '\0'};
const char msg2[] = "Test";
```

Karakter dizisi literalleri hakkında daha fazla bilgi için, “Karakter Dizisi Sabitleri” bölümüne bakın.

İfadeler İçindeki Diziler

Eğer dizi ismi, bir ifade içinde kullanılırsa (`&` ve `sizeof` operatörleri hariç), dizinin ilk elemanına işaret eden bir işaretçiymiş gibi kabul edilir. Daha fazla bilgi için “Diziler ve İşaretçiler” bölümüne bakınız.

Çok Boyutlu Diziler

Eğer bir dizi sayısal (scalar) tipte ise tek boyutludur. Tek boyutlu diziler bazen vektörler diye isimlendirilebilirler.

Çok boyutlu diziler; dizi tipinden diziler tanımlanarak oluşturulur. Bu tür diziler hafızaya öyle yerleştirilir ki, en sağdaki endeks en hızlı değişir; yani satır düzeni ile depolanırlar. Aşağıda iki boyutlu bir dizi örneği bulunmaktadır:

```
float m[ 50][ 20];  /* 50x20'lik 2 boyutlu dizi */
```

m değişkeni 50 elemanlı ve her elemanı 20 byte'lık dizi olan iki boyutlu bir dizidir. Böylece 50x20 elemandan oluşan bir matris elde ettik: İlk eleman m[0][0], en sondaki ise m[49][19]'dur, 5' inci satırın ilk elemanı m[4][0] dır.

Eğer çok boyutlu bir dizi bildiriminde diziye ilk değerler atamıyorsanız, dizinin ilk boyutunu yazmayabilirsiniz. Bu durumda dizi için başka bir yerde bellek ayrılacaktır, örneğin başka bir dosyada olabilir. Bu yöntem, dizilerin bir fonksiyonun parametresi gibi kullanıldığı zaman (yani tam boyutunu bilemediğiniz zaman) uygulanan tekniklerden biridir:

```
int a[ 3][ 2][ 4];  /*  3x2x4 lük 3 boyutlu bir dizi */

void func(int n[ ][ 2][ 4] ) {  /* ilk boyutu yazmayabiliriz */
  //...
  n[ 2][ 1][ 3] ++;  /* son elemanin arttirilmasi*/
}//~

void main() {
  //...
  func(a);
}//~!
```

Kaşlı parentezler kullanarak, çok boyutlu bir diziye uygun gördüğünüz ilk değerleri verebilirsiniz. Örnek:

```
int a[ 3][ 2]  = {{ 1,2} , { 2,6} , { 3,7}} ;
```

İşaretçiler (Pointers)

İşaretçiler, bellek adreslerini tutmak (veya işaret etmek) için kullanılan özel nesnelerdir. C dilinde, bellek içerisinde bir nesnenin adresi, tekli operatör olan (&) vasıtasıyla elde edilebilir. İşaretlenen nesneye ulaşmak için, bir işaretçi üzerinde dolaylı ulaşma operatörünü (*) kullanırız.

"A tipinde bir nesneyi işaret eden" tipte bir işaretçi A tipinde bir nesnenin adresini tutar (yani işaret eder). İşaretçiler nesne olduklarından, bir işaretçiden işaretçiye gösterme yapabilirsiniz. İşaretçiler tarafından işaret edilen diğer nesneler sıklıkla diziler, yapılar ve birliklerdir.

Bir fonksiyona işaret eden bir işaretçiyi bir kod alanında fonksiyonun çalışabilir kodlarının saklandığı bir adres olarak düşünebiliriz. Fonksiyon çağrıldığında kontrol bu adrese geçer.

İşaretçiler, işaretsiz (unsigned) tamsayıların karakteristiklerinin çoğunu içermelerine rağmen; tanımlamalar, atamalar, dönüştürmeler ve aritmetik işlemler için kendi kısıtlama ve kurallarına sahiptirler. Bundan sonraki bölümlerde bu kural ve kısıtlamalara örnekler gösterilecektir.

İşaretçi Bildirimi

İşaretçilerin bildirimi diğer değişkenler gibi yapılırlar fakat tanıtıcının önünde (*) işareti kullanılır. Bildirimin başındaki *tip* işaret edilecek nesnenin tipini belirlemektedir. Bir işaretçinin bildirimi bir tipi işaret edecek biçimde yapılmalıdır, eğer bu tip `void` olsa bile, ki bu herhangi bir nesneyi işaret edebilen bir işaretçi anlamına gelir. `void` işaretçileri sıklıkla *soysal* işaretçiler olarak da çağırılırlar ve mikroC de `char` tipi için işaretçiymiş gibi davranırlar.

Eğer tip ön tanımlı veya kullanıcı tanımlı tip ise (void tip de dahil) aşağıdaki

```
tip *p;    /* başlatılmamış işaretçi */
```

tanımlaması p'yi "*tip* için işaretçi" olarak bildirir. Henüz bildirimi yapılan *p* nesnesine tüm kapsama, süre ve görülebilirlik kuralları uygulanır. Bildirimi başka bir bakış açısıyla şöyle de düşünebiliriz : Eğer `*p, tip` türünde bir nesne ise, `p` bu tür nesneler için bir işaretçidir.

Not: İşaretçileri kullanmadan önce onlara ilk değerlerini vermelisiniz! Daha önce bildirimini yaptığımız *p işaretçisine ilk değer atamadık, o halde işaretçi henüz kullanılamaz.

Not: Çoklu işaretçi tanımlamaları durumunda, her tanıtıcı bir dolaylı erişim operatörüne ihtiyaç duyar. Örnek olarak:

```
int *pa, *pb, *pc;

/* yukaridaki satir asagidakilerle aynidir: */

int *pa;
int *pb;
int *pc;
```

Tanımlandığı halde, bir işaretçiye genellikle tekrardan başka bir tipi gösterecek şekilde değer atanabilir. mikroC işaretçilere yeniden değer atamanıza tiplemeye (typecasting) gerek kalmadan izin verir. Fakat derleyici, önceden `void` işaret edeceği bildirilmiş işaretçiler dışındaki işaretçiler için uyarı verecektir. Bir `void` işaretçiyi `void` olmayan bir işaretçiye atayabilirsiniz. Detay için "Void Tipi" bölümüne bakınız.

Null İşaretleyicileri (Hiçlik işaretleyicileri)

Bir *null işaretleyici* değeri, bir program içerisinde kullanılan geçerli herhangi bir işaretçiden farklı olduğu kesin olarak garanti edilen bir adrestir. Bir işaretçiye sayısal sabit 0 değerini atamak, o işaretçiye null işaretçi değerini atamak anlamına gelir. Sıfır yerine, NULL kısa adı (standart kütüphane başlık dosyasında tanımlanan, msl `stdio.h` gibi) okunaklılık için kullanılabilir. Tüm işaretçilere kolaylıkla sıfıra eşit veya eşit olmadıkları testi yapılabilir.

Örnek:

```
int *pn = 0;      /* bir null isaretcisi */
int *pn = NULL;   /* Bu yukaridakine esit bir tanımlamadir*/

/* Isaretciyi su sekilde test edebiliriz: */
if ( pn == 0 ) {  ... }

/* .. veya su sekilde: */
if ( pn == NULL ) {  ... }
```

Fonksiyon İşaretçileri

Fonksiyon İşaretçileri bir fonksiyonun adresini belirten işaretçiler yani değişkenlerdir.

```
// Bir fonksiyon işaretçisi tanımlamak
int (*pt2Function) (float, char, char);
```

Not: Yukaridaki şekilde açık ve net bir tanımlama yaparak değişik çağrılma stili olan (örneğin bağımsız değişkenlerinin sırası farklı, bağımsız değişkenlerin tipleri farklı veya geridönüş tipi farklı) fonksiyonların birbirleri ile uymamalarını sağlamış oluyoruz. 'Dolaylı Fonksiyon Dallanmaları' bölümüne bakın.

Bir Fonksiyon İşaretçisine bir adres atama

Bir fonksiyon işaretçisine bir fonksiyon adresi atamak çok kolaydır. Basitçe uygun ve bilinen fonksiyonun adını kullanırsınız. Fonksiyon isminin önüne adres operatörü (&) koymak isteğe bağlıdır.

```
//Bir fonksiyon isaretcisine bir adres atama.
int DoIt  (float a, char b, char c){  return a+b+c; }
pt2Function = &DoIt;  // atama
```

Örnek:

```
int addC(char x,char y){

  return x+y;
}

int subC(char x,char y){

  return x-y;
}

int mulC(char x,char y){

  return x*y;
}


int divC(char x,char y){

  return x/y;
}


int modC(char x,char y){

  return x%y;
}


// iki char tipinde bagimsiz degiskeni olan ve int donduren
// fonksiyonlar icin isaretci dizisi.

int  (*arrpf[])(char,char) = {addC, subC, mulC, divC, modC};


int res;
char i;
void main() {

   for (i=0;i<5;i++){
     res = arrpf[i](10,20);
   }

}//~!
```

İşaretçi Aritmetiği

C'de işaretçi aritmetiği şunlarla sınırlanır:

- bir işaretçiden diğerine atama,
- iki işaretçiyi mukayese etme,
- işaretçiyi sıfır (`NULL`) ile mukayese etme,
- işaretçiye tamsayı değeri ekleme/çıkarma,
- iki işaretçiyi çıkartma.

İşaretçilerin içsel olarak yapılan aritmetiği kullanılan bellek modeline ve daha baskın olan işaretçi değiştiricilerinin varlığına bağlıdır. İşaretçiler ile aritmetik işlem yaparken işaretçilerin bir nesneler dizisine işaret ettikleri varsayılır.

Diziler ve İşaretçiler

C'de, diziler ve işaretçiler tam anlamıyla bağımsız tipler değildirler. Dizinin ismi ifade içerisinde değerlendirileceği vakit (& ve sizeof operatörleri ile hariç), dizi ismi dizinin ilk elemanını gösteren bir işaretçiye dönüştürülür. Bundan dolayı, diziler modifiye edilebilen ldeğerleri değildir.

Parantezler`[ ]` dizinin indekslerini gösterir. Hemen alttaki ifade

```
id[ exp]
```

aşağıdaki

```
*((id) + (exp))
```

ifadesine eşdeğer olarak tanımlanır:

id bir işaretçidir ve *exp* bir tamsayı, veya
id bir tamsayıdır ve *exp* bir işaretçi.

Aşağıdaki gösterim doğrudur:

```
&a[ i]   =  a + i
 a[ i]   =  *(a + i)
```

Bu bilgilerin kılavuzluğunda, şöyle yazabiliriz:

```
pa = &a[4];        // pa, a[4]'e işaretçi
x = *(pa + 3);     // x = a[7]
y = *pa + 3;       // y = a[4] + 3
```

Aynı zamanda, operatörlerin işlem önceliklerine dikkat etmelisiniz:

```
*pa++;       // *(pa++)'ye eşittir, işaretçiyi arttırma!
(*pa)++;     // işaretçi nesnesini arttırma!
```

Aşağıdaki ifadeler de geçerlidir, fakat bu tür söz diziminden kaçınmak en iyisidir çünkü, kodu gerçekten okunaksız ve anlaşılması çok zor bir hale getirebilir :

```
(a + 1)[i] = 3;
// şuna benzerdir : *((a + 1) + i) = 3, veya a[i + 1] = 3

(i + 2)[a] = 0;
// şuna benzerdir: *((i + 2) + a) = 0, veya a[i + 2] = 0
```

Atama ve Karşılaştırma

Eğer işaretçilerin tipleri aynı ise bir işaretçinin değerini diğerine eşittir (=) operatörünü kullanarak kolayca atayabilirsiniz. Eğer farklı tipten iseler, tip çevrimi (`typecast`) operatörünü kullanmalısınız. Eğer işaretçilerden biri soysal (generic) ise açıkça yapılan tip dönüşümüne gerek yoktur.

Bir işaretçiye 0 sabit değeri atanırsa ilgili işaretçiye bir hiçlik (null) değeri atanır. Anımsatıcı isim olan `NULL` (standart kütüphane başlık dosyalarında tanımlı, `stdio.h` gibi) okunaklılık için kullanılabilir.

Aynı diziyi gösteren iki işaretçi ilişkisel operatörler (==, !=, <, <=, >, ve >=) kullanılarak karşılaştırılabilir. Bu işlemlerin sonuçları, sanki karşılaştırma ifadesi içinde dizi elemanlarının indekslerinin kullanılmış olması gibidir ve hatta aynıdır. Örneğin:

```
int *pa = &a[4], *pb = &a[2];

if (pa > pb) {   ...
  // 4 buyuktur 2 oldugu icin bu kod kismi calistirilacak
}
```

Aynı zamanda işaretçileri sıfır değeri ile mukayese edebilirsiniz. Bu işaretçinin gerçekten herhangi bir şeyi işaret edip etmediğini test etmek içindir. Tüm işaretçiler sıfır (NULL) ile eşitlik veya eşitsizlik için test edilebilirler:

```
if (pa == NULL) {  ... }
if (pb != NULL) {  ... }
```

Not: Farklı nesneler/diziler gösteren işaretçileri karşılaştırmak programlayıcının sorumluluğu dahilinde yapılabilir; fakat, verinin fiziksel depolama yapısına çok iyi hakim olmak gereklidir.

İşaretçilerde Toplama İşlemi

+, ++, ve += operatörlerini bir tamsayı değeri bir işaretçiye eklemek için kullanabilirsiniz. Toplamanın sonucu, sadece eğer işaretçi bir dizinin bir elemanını gösteriyorsa ve eğer sonuç aynı diziyi (veya hemen sonrasını) gösteren bir işaretçi ise tanımlıdır.

Eğer bir işaretçi örneğin bir X *tipine* işaret için tanımlı ise, bir tamsayı değerin işaretçiye eklenmesi o tipin nesnelerinin o kadar sayısı kadar işaretçiyi ilerletir. Pratik düşünürsek, `P+n`, `P` işaretçisini (`n*sizeof(tip)`) bayt kadar ilerletir. (Tabi işaretçi geçerli aralıkta yani ilk eleman ile son elemanın bir ötesinde kalmalıdır). Eğer *tip* 10 byte boyuta sahipse, *tip* işaretçisine 5 eklendiğinde bellek içerisinde işaretçi 50 byte ilerler. `void` tip olduğu vakit, adım boyutu bir byte'tır.

Örnek olarak:

```
int a[10];          // dizi 10 tane int eleman iceriyor
int *pa = &a[0];    // pa,int icin isaretci, a[0]'i isaret ediyor

*(pa + 3) = 6;      // pa+3, a[3]'u gosteren bir isaretci,
                    // bu nedenle a[3] su anda 6'ya esittir
pa++; /* pa simdi dizinin bir sonraki elemanini yani a[1]'i
isaret ediyor */
```

"Son elemandan bir sonraki eleman" diye bir eleman yoktur, fakat işaretçinin böyle bir değer almasına izin verilir. C bir işaretçi toplama işleminin sonucunun tanımlı olmasını sonuç diziden bir sonraki elemanı işaret etse bile garantiler. Eğer `P` en sonuncu dizi elemanı gösteriyorsa, `P+1` geçerlidir, fakat `P+2` tanımsızdır.

Bu size işaretçinin arttırılması yoluyla dizi elemanlarına bir düzen içerisinde erişen döngüleri yazmanıza izin verir. En son yinelemede dizinin son elemanından bir sonraki eleman için bir işaretçiye sahip olacaksınız ki bu geçerlidir. Buna rağmen, "son elemandan bir sonraki eleman"a dolaylı erişim operatörü (*) ile erişmek tanımlanmamış davranışa yol açar.

Örnek olarak::

```
void f (some_type a[], int n) {
     /*  f fonksiyonu a dizisinin elemanlarini isler; */
     /* a dizisi some_type türünden n elemana sahiptir */

  int i;
  some_type *p = &a[0];

  for (i = 0; i < n; i++) {
             /* .. burada *p ile birseyler yaptik .. */
    p++;     /* .. ve son yinelemede p, a dizisinin son
                    elemanini gecti */
  }
  /* bu noktada, *p tanimsizdir ! */
}
```

İşaretçilerde Çıkarma İşlemi

Toplamaya benzerdir. -, --, ve -= operatörlerini bir tamsayı değeri bir işaretçiden çıkarmak için kullanabilirsiniz.

Aynı zamanda, bir işaretçiyi bir işaretçiden çıkarabilirsiniz. Fark iki işaretli adres arasındaki mesafenin eleman sayısı olarak değeridir.

Örnek olarak:

```
void main() {
  int i=0;
  int a[10];
  int * pi1;
  int * pi2;

  pi1= &a[0];  // sifirinci eleman
  pi2= &a[4];  // dorduncu eleman
  // isaretci aritmetigine gore cikarma islemindeki
  // fark aradaki eleman sayisi cinsinden hesaplanir.
  i = pi2-pi1; // i = 4 olur
}
```

Yapılar (Structures)

Bir yapı, isimlendirilmiş üyelerinin (veya bileşenlerinin) bir kullanıcı-tanımlı toplamını gösteren bir türetilmiş tiptir. Üyeler herhangi bir sırada, herhangi bir tipten, temel veya türetilmiş olabilir. Ek olarak, bir yapı üyesi başka hiçbir yerde izin verilmeyen bir bit alan tipi olabilir.

Dizilerden farklı olarak, yapılar tek bir nesne olarak kabul edilirler. mikroC yapı tipi, karmaşık veri yapılarını tek değişkenler kadar kolay işlemenize olanak sağlar.

Not: mikroC anonim yapıları desteklemez (ANSI standardından farklı olarak).

Yapı tanımlama ve ilk değer verme

Yapılar `struct` anahtar kelimesi kullanılarak yapılandırılır:

```
struct etiket { üye-tanımlayıcı-liste };
```

Burada, *etiket* yapının ismidir. *üye-tanımlayıcı-liste* yapı üyelerinin bir listesidir, gerçek anlamda bir değişken tanımları listesidir. Yapılandırılmış tiplerin değişkenleri herhangi başka bir tipin değişkenleri gibi tanımlanırlar.

Yapının üyesinin tipi, ilgili yapının kendisinin tipi ile aynı olamaz. Bununla beraber aşağıdaki örnekte olduğu gibi, bir üye, tanımlanan yapı için bir işaretçi olabilir:

```
struct mystruct { mystruct s;};         /* gecersiz! */
struct mystruct { mystruct *ps;};       /* gecerli */
```

Aynı zamanda bir yapı, bildirilen yapının bir örneğini tanımlarken, içinde önceki tanımlı yapı tiplerini bulundurabilir. Aşağıda bununla ilgili bir örnek mevcuttur:

```
/* yapi bir nokta tanimliyor:  */
struct nokta {float x, y;};

/* yapi bir cember tanimliyor: */
struct cember {
  double r;
  struct nokta merkez;
} o1, o2; /* cember tipinden o1 ve o2 degiskenlerini tanimlar */
```

Unutmayın ki yapı etiketlerini es geçebilirsiniz, fakat bu durumda artık başka yerde bu tipten başka nesneleri tanımlayamazsınız. Daha fazla bilgi için, "Etiketlenmemiş Yapılar"a bakınız.

Yapılara ilk değerleri, dizilere benzer olarak kaşlı parantezler içerisinde virgül ile sınırlandırılmış değerler dizisi atanarak verilir. Bir önceki örnekten devam edersek:

```
/* p ve q noktalarini bildir ve ilk deger ver:  */
struct Dot p = {1., 1.}, q = {3.7, -0.5};

/* o2 ve o3 cemberlerini bildir ve ilk deger ver: */
struct cember o2 = {1, {0, 0}};       // r=1, merkez (0, 0) da
struct cember o3 = {4, {  1.2, -3 }}; // r=4, merkez (1.2, -3) da
```

Eksik Bildirimler

Eksik tanımlama aynı zamanda ileri tanımlama olarak da bilinir. Bir `A` yapı tipi için bir işaretçi, `A` yapısı tanımlanmadan önce başka bir `B` tanımı içerisinde geçerli bir şekilde kullanılabilir:

```
struct A;     // eksik
struct B {struct A *pa;};
struct A {struct B *pb;};
```

`A` nın ilk görünüşüne eksik bildirim denir. Çünkü, o noktada onun için herhangi bir tanımlama yoktur. Bir eksik bildirime burada izin verilir. Çünkü, B nin tanımı A'nın boyutuna ihtiyaç duymaz.

Etiketsiz Yapılar ve Typedef'ler

Eğer yapı etiketini es geçerseniz, etiketsiz bir yapı elde etmiş olursunuz. Etiketsiz yapıları virgülle ayrılmış `yapı-kimlik-listesi` içindeki tanıtıcıları bildirmek için kullanabilirsiniz. Böylece bu yapıdan veya türevi tanıtıcıları bildirebilirsiniz. Fakat diğer hiçbir yerde bu tipten başka nesneler tanımlayamazsınız.

Bir yapı tanımlarken etiket kullanarak veya kullanmaksızın bir typedef (yani yeni bir tip tanımı) oluşturmak mümkündür:

```
typedef struct {  ... } Mystruct;
Mystruct s, *ps, arrs[10];
```

Yapı Atamaları

Aynı tipten yapıların değişkenleri birinden diğerine basitçe atama operatörü (=) kullanarak atanabilir. Bu işlem yapının bütün içeriğini diğer yapıya kopyalayacaktır ve burada yapının ne kadar karmaşık olduğunun bir önemi yoktur.

Şuna dikkat ediniz ki, yapı cinsinden iki değişkenin aynı yapı türünden olması için ya aynı komut ile tanımlanmış olmaları ya da tanımlanırken aynı tip tanıtıcısının kullanılmış olması gereklidir. Bir örnekle daha iyi açıklayabiliriz:

```
/* a ve b aynı tiptendirler: */
struct {int m1, m2;} a, b;

/* fakat c ve d her ne kadar yapi tanimlari ozdes olsa da ayni
tipten degildirler: */
struct {int m1, m2;} c;
struct {int m1, m2;} d;
```

Yapının Boyutu

Yapının bellek içerisindeki boyutunu `sizeof` operatörü sayesinde elde edebilirsiniz. Yapının boyutu üyelerinin boyutlarının toplamına eşit olmak zorunda değildir. Genellikle de bellek yerleşimi sınırlamalarından dolayı daha büyüktür.

Yapılar ve Fonksiyonlar

Bir fonksiyon yapı tipi veya bir yapı tipi işaretçisi döndürebilir:

```
mystruct func1();   // func1() bir yapi dondurur.
mystruct *func2();  // func2() yapiya bir isaretci dondurur
```

Bir yapı bir fonksiyona aşağıdaki yollarla bir bağımsız değişken olarak geçebilir:

```
void func1(mystruct s);          // dogrudan
void func2(mystruct *sptr);      // isaretci yoluyla
```

Yapı Üyesine Erişim

Yapı ve birlik üyelerine aşağıdaki iki seçim operatörü ile erişilir:

. (nokta)
-> (sağ ok)

'.' operatörü doğrudan üye seçici olarak isimlendirilir ve yapının üyelerinden birine doğrudan ulaşmak için kullanılır. Varsayalım ki s nesnesi, S yapı tipindendir. Eğer m, s yapısı içinde tanımlı M tipinden bir üye tanıtıcısı ise aşağıdaki ifade,

```
s.m   //  m uyesine dogrudan erisim
```

M tipindendir ve s içerisindeki m üye nesnesini gösterir.

'-> ' operatörü dolaylı (veya işaretçi) üye seçici olarak bilinir. Varsayalım ki ps , s için bir işaretçidir. Eğer m, s yapısı içinde tanımlı M tipinden bir üye tanıtıcısı ise, aşağıdaki ifade

```
ps->m //  m uyesine dolayli erisim;
      // (*ps).m' ya ozdes
```

M tipindendir ve s içerisindeki m üye nesnesini gösterir. `ps->m` ifadesi `(*ps).` ifadesi için uygun bir kısa gösterimdir.

Örnek olarak:

```
struct mystruct {
  int i; char str[10]; double d;
} s, *sptr = &s;
    .
    .
    .
s.i = 3;           // mystruct s'in i uyesine deger atama
sptr -> d = 1.23;  // mystruct s'in d uyesine deger atama
```

`s.m` ifadesi şu koşullar altında bir sol-değerdir (lvalue) : s bir sol-değer olmalıdır ve m bir dizi tipi olmamalıdır. Benzer şekilde `sptr->m` ifadesi ancak m bir dizi tipi değilse bir sol-değerdir.

İç-içe geçmiş Yapılara Erişim

Eğer B yapısı, tipi A yapısı olan bir alan içerirse, A'nın üyelerine üye seçicilerin iki kez uygulanmasıyla erişilebilir:

```
struct A {
  int j; double x;
};
struct B {
  int i; struct A a; double d;
}  s, *sptr;

//...

s.i = 3;            // B nin i üyesine 3 atama
s.a.j = 2;          // A nın j üyesine 2 atama
sptr->d = 1.23;     // B nin d üyesine 1.23 atama
sptr->a.x = 3.14;   // A nin x üyesine 3.14 atama
```

Yapı Tekliği

Her yapı bildirimi bir tek yapı tipini tanıtır. Şöyle ki

```
struct A {
  int i,j; double d;
}  aa, aaa;

struct B {
  int i,j; double d;
}  bb;
```

aa ve aaa nesneleri her ikisi de A yapısının tipindedirler. Fakat aa ve bb nesneleri farklı yapı tiplerindedirler. Yapılar, ancak kaynak ve hedefleri aynı tipte ise atanabilirler:

```
aa = aaa;     /* tamam: ayni tip, uye-uye atama */
aa = bb;      /* gecersiz: farkli tipler */

/* fakat uye-uye atama gerceklestirebilirsiniz: */
aa.i = bb.i;
aa.j = bb.j;
aa.d = bb.d;
```

Birlikler (Unions)

Birlik tipleri yapı tiplerinin söz dizimsel ve fonksiyonel özelliklerinin çoğunu paylaşan türetilmiş tiplerdir. Esas farklılık şudur: birlikler herhangi bir zamanda yalnızca en yakın zamanda değişen üyesinin "aktif" olmasına izin verir.

Not: mikroC anonim birlikleri desteklemez (ANSI standardından farklı olarak).

Birlik Bildirimi

Birlikler aynen yapılar gibi ancak `struct` anahtar kelimesi yerine `union` kullanılarak tanımlanırlar:

```
union etiket { uye-tanitici-listesi };
```

Yapı üyelerinden farklı olarak, bir zamanda birlik üyelerinden yalnızca birinin değeri birlik içerisinde saklanabilir. Aşağıdaki basit örneğe bakalım:

```
union myunion {   // birligin etiketi 'myunion'
  int i;
  double d;
  char ch;
} mu, *pm = &mu;
```

`myunion` birliği tipinden `mu` tanıtıcısı ile tanımlanmış birliğin içerisinde 2 bayt'lık bir `int`, veya 4 bayt'lık bir `double`, veya tek bayt'lık bir `char` tutmak mümkündür, ancak bir zamanda diliminde sadece bir tanesi tutulabilir.

Birlik Boyutu

Bir birliğin boyutu, onu oluşturan üyelerinden en geniş boyuta sahip olan üyesinin boyutudur. Önceki örneğimizde, hem `sizeof`(*myunion* birliği) hemde `size of`(*mu*) 4 döndürür. Fakat mu `int` nesne tuttuğu zaman 2 byte kullanılmaz. Ve *mu* bir `char` tuttuğu zaman ise 3 byte kullanılmaz.

Birlik Üyesine Erişim

Birlik üyeleri, yapı üye seçicileri ile erişilebilirler (`.` and `->`). Fakat dikkatli olmak gereklidir. Bir sonraki sayfadaki örneğe bakınız.

Bir önceki örnekteki bildirimleri de göz önüne alırsak:

```
mu.d = 4.016;
Lcd_Out_Cp(FloatToStr(mu.d));     // Tamam: mu.d = 4.016 goruntulenir
Lcd_Out_Cp(IntToStr(mu.i));       // Beklenmedik bir sonuc!!!

pm->i = 3;
Lcd_Out_Cp(IntToStr(mu.i));       // Tamam: mu.i = 3 goruntulenir
```

İkinci `Lcd_Out_Cp` kurallara göre geçerlidir çünkü `mu.i` bir tamsayı tiptir. Ancak, `mu.i` içerisindeki bit örneği (bit pattern) daha önce atanan double parçalarından alınmıştır. Bu yüzden de büyük olasılıkla işe yarar bir tamsayı sağlamayacaktır.

Doğru şekilde kullanıldığında bir birliği gösteren bir işaretçi bu birliğin tüm elemanlarına ulaşabilir ve bunun tam tersi de geçerlidir.

Bit Alanları

Bit alanları belli sayıda bit'in oluşturduğu gruptur (Bağlı oldukları bir tanımlayıcı olsun veya olmasın). Bit alanları, yapıları (struct'ları) kullanıcı tanımlı alt parçalara ayırmada bize bir yol sağlarlar.

Yapılar ve birlikler bit alanları içerebilirler. Bit alanları 16 bit'e kadar olabilir.

Bir bit alanının adresini edinmek olanak dışıdır.

Not: Eğer 8-bit değişkenlerin (char ve unsigned short) veya yazmaçların belirli bitlerini işlemek istiyorsanız, bit alanı tanımlamanıza gerek yoktur. Daha uygun ve şık bir çözüm, mikroC'nin tek tek bitlere erişim özelliğini kullanmak olacaktır. Daha fazla bilgi için "Tek-tek Bitlere Erişim" bölümüne bakın.

Bit Alanları Bildirimi Yapmak

Bit alanları sadece yapılar içerisinde bildirilebilir. Bir yapıyı normal bir şekilde bildiriniz ve tek tek bit alanlarını da aşağıdaki gibi atayınız (alanlar *unsigned*(işaretsiz) olmalıdırlar):

```
struct etiket {  unsigned bitalani-tanitici-listesi; }
```

Burada *etiket (tag),* yapının isteğe bağlı olan ismidir. *bitalanı_bildirici_listesi* bit alanlarının bir listesidir. Her bileşen tanıtıcısı için bir adet "iki nokta üst üste işareti" ve açık bir şekilde genişlik bilgisine ihtiyaç vardır. Toplam bileşenlerin genişliği iki byte'ı geçmemelidir (16 bit).

Bir nesne olarak bit alanlarından oluşan bir yapı iki byte alabilir. Bu durumda tek-tek alanlar sağdan sola iki byte olarak paketlenirler. *bitalanı-tanıtıcı listesi*'nde, tanıtıcı(lar)ı es geçerek yapay "dolgu" alanları belirleyebilirsiniz. Böylelikle ilgilenmediğiniz bitleri atlamış olursunuz.
Örnek olarak, yazmacın sadece 2-4 bitlerini blok olarak işlemek istiyorsak şöyle bir yapı oluşturabiliriz:

```
struct {
  unsigned  : 2,    // 0 ve 1 bitleri atlandi, burada tanitici yok
  mybits    : 3;    // ilgilendigimiz bitler 2, 3, ve 4
// 5, 6 ve 7 inci bitler gizli olarak bos birakildi.
} myreg;
```

örnek:

```
typedef struct {
  prescaler  : 2; timeronoff : 1; postscaler : 4;} mybitfield;
```

Bu örnek `mybitfield` yapısı ve üç bileşenini tanımlar: `prescaler` (0 ve 1 bitleri), `timeronoff` (bit 2), ve `postscaler` (3, 4, 5,ve 6 bitleri).

Bit Alanlarına Erişim.

Bit alanları yapı üyelerindeki ile aynı yolla erişilebilirler. Doğrudan veya dolaylı seçici operatörlerini kullanınız. (`.` ve `->`). Örnek olarak, biraz önce tanımladığımız `mybitfield` üzerinde çalışabiliriz:

```
//  TimerControl diye bir bit alani tanimla:
mybitfield TimerControl;

void main() {
  TimerControl.prescaler  = 0;
  TimerControl.timeronoff = 1;
  TimerControl.postscaler = 3;
  T2CON = TimerControl;
}
```

TİP DÖNÜŞÜMLERİ (TYPE CONVERSIONS)

C, kesin olarak tiplendirilmiş bir dildir, yani her operatör, deyim ve fonksiyon uygun olarak tipleri verilmiş operandlar/bağımsız değişkenler ister. Buna karşın, çoğu kez ifadelerde "uyuşmayan" tipler kullanırız. Bu durumda, tip dönüştürümlerine ihtiyaç duyulur. Bir tipten nesneyi dönüştürme, nesne aynı kaldığı halde tipinin diğer tipe dönüşmesidir. (Örnekle açıklarsak : Bir nesneye farklı bir tipi uygulamak). C dili yerleşik tipler için, gerekli olduğunda derleyici tarafından sağlanan birtakım standart dönüşümler tanımlar.

Dönüştürmeye aşağıdaki durumlarda ihtiyaç duyulur:

- Eğer bir deyim (statement) özellikle bir tipten ifadeye (expression) ihtiyaç duyuyorsa (dil tanımlamasına göre) ve biz farklı tipten bir ifade (expression) kullanıyorsak,
- eğer bir işlev (operatör) özellikle bir tipten işlenene (operand) ihtiyaç duyuyorsa ve biz farklı bir tip işlenen (operand) kullanıyorsak,
- eğer bir fonksiyon özellikle bir tipten bir formal parametreye ihtiyaç duyuyorsa ve biz farklı bir tipin nesnesini geçiriyorsak,
- eğer bir fonksiyondaki "return" anahtar kelimesinden sonra kullandığımız ifade fonksiyonun bildirilen dönüş tipi ile uyuşmuyorsa,
- eğer bir nesneye bildirim sırasında farklı tipten bir ilk değer atıyorsak.

Bu durumlarda, derleyici (kullanıcının müdahalesine gerek kalmadan) otomatik ve içsel olarak tip dönüşümü sağlar. Aynı zamanda, kullanıcı `typecast` operatörü yardımıyla açıkça belirtilmiş bir dönüşüm talep edebilir. Bu konuda daha fazla bilgi için "Açıkça Tip Dönüştürme - Typecasting" bölümüne bakın.

Standart Dönüşümler

Standart dönüşümler C içerisinde zaten mevcuttur. Bu dönüşümler, program içerisinde ihtiyaç duyulduğunda otomatik olarak çalıştırılırlar. Ayrıca, `typecast` operatörü vasıtasıyla açıkça belirtilerek kullanılabilirler.

Otomatik (içsel) dönüştürmenin esas kuralı şudur: Bir ifade içerisindeki basit tipteki "işlenen" (operand) diğer daha karmaşık olan "işlenen" tipine dönüştürülür (yani terfi ettirilir). Daha sonra oluşan sonuç tipi, daha karmaşık olan "işlenen" tipindedir.

Aritmetik Dönüşümler

Bir aritmetik ifade kullandığınız zaman, mesela a+b olsun ve a ve b farklı aritmetik tiplerden olsun, bu durumda mikroC bu ifadeyi değerlendirmeden önce içsel (implicit) tip dönüştürümü gerçekleştirir. Bu standart dönüşümler doğruluk ve tutarlılık çerçevesinde "en düşük" tiplerden "en yüksek" tiplere yükselmeyi içerir.

İşaretli karakter (`signed char`) türünden bir nesnenin (mesela bir değişkenin) tümleşik nesnelere atanması otomatik işaret eklenmesi ile sonuçlanır. `int` tipine dönüştürüldüklerinde; `signed char` tipinin nesneleri daima işaret ekini kullanırlar ve `unsigned char` tipinin nesneleri ise üst baytı (high byte) sıfıra çekerler.

Daha uzun bir tümleşik tipin daha kısa bir tipe dönüştürülmesinde üst bitler kırpılır ve alt bitleri değiştirmeden bırakılır. Kısa bir tümleşik tipin daha uzun bir tipe dönüştürülmesinde, yeni değerin ekstra bitlerine, işaret uzantıları veya sıfırlar doldurulur. Bu işlem kısa tipin işaretli olup olmamasına bağlıdır.

Not: Kayan noktalı verinin tamsayı değere dönüştürülmesinde (atamalarda veya açık typecast yoluyla) eğer `float` değer hedef tümleşik tipin kapsamını aşmazsa doğru sonuçlar üretilir.
Aşağıda mikroC'nin aritmetik ifadelerde dönüştürme için kullandığı adımlar vardır. Önce herhangi küçük bir tümleşik tip aşağıdaki kurallara göre dönüştürülür:

1. `char` , `int'e dönüşür`
2. `signed char` , `int'e dönüşür`, aynı değerle
3. `short` , `int'e dönüşür`, aynı değerle işaret uzantılı bir şekilde
4. `unsigned short`, unsigned int'e dönüşür, aynı değerle, sıfır doldurularak
5. `enum` , `int'e dönüşür`, aynı değerle

Bundan sonra, bir operatörle işlenen iki değer ya `int`'tir (`long` ve `unsigned` dahil) yada `float`'tır (mikroC de `double` ve `long double`'a eşdeğer).

1. Öncelikle, eğer herhangi bir işlenen (operand) `float` ise, diğer işlenen `float`'a dönüştürülür,
2. Değilse, eğer herhangi bir işlenen `unsigned long` ise, diğer işlenen de `unsigned long`'a dönüştürülür.
3. Değilse, eğer herhangi bir işlenen `long` ise diğer işlenen de `long`'a dönüştürülür.
4. Değilse, eğer herhangi bir işlenen `unsigned` ise, diğer işlenen de `unsigned`'a dönüştürülür.
5. Aksi halde her iki işlenen de zaten `int` türündendir.

İfadenin (expression) sonucu her iki operandın tipleri ile aynıdır.

Aşağıda birkaç içsel dönüşüm örneği mevcuttur:

```
2+3.1        // = 2. + 3.1 = 5.1
5/4*3.       // = (5/4)*3. = 1*3. = 1.*3. = 3.0
3.*5/4       // = (3.*5)/4 = (3.*5.)/4 = 15./4 = 15./4. = 3.75
```

İşaretçi Dönüşümleri

İşaretçi tipleri, tip dönüştürme mekanizmasını kullanarak diğer işaretçi tiplerine dönüştürülebilir:

```
char *str;
int *ip;
str = (char *)ip;
```

Daha genel olarak, (*type**) ifadesi bir işaretçiyi "ilgili *type* tipine bir işaretçi"ye dönüştürecektir.

Açıkça Belirtilen Tip Dönüşümleri (Typecasting)

Çoğu durumda, ihtiyaç duyulan yerlerde, kullanıcı müdahalesine gerek kalmadan, derleyici bir otomatik içsel tip dönüştürmesi sağlayacaktır. Ancak siz de, dönüştürme operatörü kullanarak bir operandı diğer bir tipe açık (explicitly) bir şekilde dönüştürebilirsiniz:

Örnek:

```
char a, b;

/* Asagidaki satir a'yi isaretsiz int yapacaktir: */

(unsigned int) a;

/* Asagidaki satir a yi double yapacaktir,
   daha sonra b yi de otomatik olarak double yapacaktir,
   sonuc double tip degerinde olacaktır: */
(double) a + b;     //Bu ifade ((double) a) + b  'ye esittir;
```

BİLDİRİMLER (DECLARATIONS)

Bildirimler ile İlgili Ön Bilgi

Bildirimler, bir programa bir ya da daha fazla isim tanıtır. Derleyiciyi; ismin ne olduğu, tipinin ne olduğu, işlemlerin onunla neye izin verildiği, v.s. hakkında bilgilendirir. Bu bölüm bildirimler ile ilgili konuları inceleyecektir, bunlar: bildirimler, tanımlamalar, bildirim belirteçleri ve ilk değer vermedir.

Bildirilen nesneler şunlar olabilir:

- Değişkenler
- Sabitler
- Fonksiyonlar
- Tipler
- Yapı, birlik ve numaralama etiketleri
- Yapı üyeleri
- Birlik Üyeleri
- Değişik tiplerin dizileri
- Deyim etiketleri
- Ön-işlemci makroları

Bildirimler ve Tanımlar (Declarations and Definitions)

Bildirimleri tanımlamak, aynı zamanda *tanımlamalar (definitions)* olarak da bilinir; bir nesnenin bildiriminin yanında, aynı zamanda nesnenin oluşumunu da (yer ve zaman) kurar. Yani, fiziksel bellek yerleşimi ve olası ilk değer atama yapılır. Başvuru bildirimleri veya sadece bildirimler (declarations) basitçe onların tanıtıcılarını ve tiplerini derleyici için bilindik yaparlar.

Gözden geçirecek olursak, bildirimler aşağıdakiler hariç aynı zamanda birer tanımlayıcılardır:

- Bir fonksiyonu gövdesini belirtmeden bildirir,
- harici (extern) bir belirtece sahiptir ve herhangi bir ilk değer atayıcısı veya gövdesi yoktur (Fonksiyon olduğu durumda),
- bir typedef bildirimidir.

Aynı tanıtıcı için birçok başvuru bildirimi olabilir, özellikle de çok dosyalı programlarda, fakat bu tanıtıcı için sadece bir tane tanımlayıcı bildirime izin verilir.

Örnek:

```
/* Burada max fonksiyonunun tanımlama yapmayan bir bildirimi */
/* vardir ve sadece derleyiciye max in bir fonksiyon oldugu */
/* bilgisini veriyor */
int max();

/*  Ve iste max fonksiyonunun tanimi: */
int max(int x, int y) {
  return (x>=y) ? x : y;
}

int i;  /* i degiskeninin tanimi */
int i;  /* hata durumu: i zaten onceden tanimlidir! */
```

Bildirimler ve bildiriciler

Bir bildirim bir isimler listesidir. Bu isimler bazen bildiriciler veya tanıtıcılar olarak adlandırılırlar. Bildirim seçimlik depolama sınıfı belirteci, tip belirteci ve diğer değiştiriciler ile başlar. Tanıtıcılar virgüllerle ayrılırlar ve liste noktalı virgül ile sonlandırılır.

Değişken tanıtıcılarının bildirimleri aşağıdaki düzene sahiptir:

```
depolama-sınıfı [ tip-niteleyici] tip var1 [ =init1], var2 [ =init2],
...;
```

burada `var1`, `var2`,... seçimlik bir ilk değer ile birlikte farklı tanıtıcıların herhangi bir şekilde sıralanışıdır. Her değişken `tip` olacak şekilde bildirilir. Eğer `tip` es geçilirse `tip, int` olarak varsayılır. `depolama-sınıfı` belirleyicisi `extern`, `static`, `register`, veya varsayılan `auto` değerlerini alabilir. Opsiyonel olan *tip-niteleyici,* `const` veya `volatile` değerlerini alabilir. Daha fazla bilgi için" Depolama Sınıfları ve Tip Niteleyiciler" bölümlerine bakınız.

Aşağıda değişken bildiriminin bir örneği yer almaktadır:

```
/*  x, y, ve z isminde 3 değişken olustur ve x ve y degiskenlerine
sırasıyla 1 ve 2 ilk degerlerini ver. */
int x = 1, y = 2, z;   // z ilk deger verilmemis olarak kalir
```

Bunların hepsi tanımlayıcı bildirimlerdir; depolama alanı ayrılmıştır ve seçimlik ilk değerler uygulanmıştır.

Bağlantı (Linkage)

Bir çalıştırılabilir program genellikle bağımsız birkaç çevrim ünitesinin derlenmesi ve daha sonra da oluşan object dosyalarının önceden var olan kütüphane dosyaları ile bağlanması ile oluşturulur. Çevrim ünitesi (veya çevirisi yapılacak ünite) içerilecek dosyaları eklenmiş ve ön-işlemci direktifleri ile çıkarılması istenen kısımları çıkarılmış olan bir çeşit kaynak dosyasıdır. Aynı tanıtıcı farklı kapsamlar (örneğin dosyalar) içerisinde bildirildiğinde veya aynı kapsamda birden fazla kez bildirildiğinde sorunlar ortaya çıkar.

Bağlantı (linkage) bir tanıtıcının her örneğinin özel bir nesne veya fonksiyon ile doğru bir şekilde ilişkilendirilmesini sağlayan bir süreçtir. Tüm tanıtıcılar, kendi kapsamlarına da yakından ilgili olarak, iki bağlantı özniteliğinden birine sahiptirler: Dışsal bağlantı (External Linkage) veya içsel bağlantı (Internal Linkage). Bu öz nitelikler sizin bildirimlerinizin yer ve biçimine göre ve depolama sınıfı belirteçleri olan `static` veya `extern` kelimelerinin açıkca veya içsel kullanımı ile belirlenir.

Harici bağlantılı olan bir tanıtıcının her örneği programı oluşturan tüm dosyalar ve kütüphaneler gözönüne alındığında aynı nesne veya fonksiyonu temsil eder. Dahili bağlantılı olan bir tanıtıcının her örneği, sadece tek bir dosyada aynı nesne veya fonksiyonu temsil eder.

Bağlantı Kuralları

Yerel (local) isimler içsel bağlantıya sahiptirler; aynı tanıtıcı farklı dosyalarda farklı nesneleri göstermek için kullanılabilirler. Global isimler harici bağlantıya sahiptirler; tanıtıcı tüm program boyunca aynı nesneyi gösterir.

Eğer aynı tanıtıcı aynı dosya içerisinde hem dışsal hemde içsel bağlantı ile ortaya çıkarsa, tanıtıcı içsel bağlantıya sahip olacaktır.

İçsel Bağlantı Kuralları:

1. Dosya kapsamına (scope) sahip ve açıkça `static` olarak bildirilmiş isimler, içsel bağlantıya sahiptir.
2. Dosya kapsamına sahip, açıkça `const` olarak bildirilmiş ve özellikle açıkça `extern` olarak bildirilmemiş isimler içsel bağlantıya sahiptirler.
3. `typedef` isimleri (tip tanımları) içsel bağlantıya sahiptirler,
4. Numaralama (enumeration) sabitleri içsel bağlantıya sahiptirler.

Dışsal Bağlantı Kuralları:

1. Dosya kapsamına sahip ve az önce anlatılan içsel bağlantı kurallarına uymayan tüm isimler, dışsal bağlantıya sahiptirler.

Depolama sınıfı tanıtıcıları olan `auto` ve `register`, dışsal bir bildirimde yer alamazlar. Bir çevrim ünitesindeki içsel bağlantı ile bildirilmiş her tanıtıcı için birden fazla dışsal tanım verilemez. Dışsal bir tanımlama bir nesne veya fonksiyonu tanımlayan dışsal bir bildirimdir; aynı zamanda depolama alanı ayırır. Eğer dışsal bağlantı ile bildirilmiş bir tanıtıcı bir ifadede kullanılırsa (`sizeof` komutunun işleneni veya işlenenin bir parçası hariç), bu durumda tüm program içinde biryerlerde bu tanıtıcının sadece bir dışsal tanımı bulunmalıdır.

mikroC dışsal isimlerin daha sonradan tekrar bildirilmesine izin verir. Böylece örneğin diziler, yapılar ve birlikler için önceki bildirimlerden daha fazla bilgi taşıyan bildirimler yapılabilir. Örnek:

```
int a[];                  // Bu bildirimde boyut yok.

struct mystruct;          // Bu bildirimde sadece etiket var,
                          // herhangi bir uye bildirimi yok.
  .
  .
  .

int a[3] = {1, 2, 3};     // Simdi boyut ve ilk degerler
                          // saglaniyor.

struct mystruct {
  int i, j;
};                        // Simdi uye bildirimleri yapiliyor.
```

Depolama Sınıfları

Tanıtıcıların nesneler ile ilişkilendirilmesinde her tanıtıcının en azından iki özniteliğe sahip olması gerekir: depolama sınıfı ve tip (bazen veri tipi olarak da kullanılır). mikroC derleyicisi bu özellikleri kaynak kod içerisindeki örtük (kendinden) veya açıkça yapılmış bildirimlerden anlar.

Depolama sınfı nesnenin yerleşimini (veri kesimi (data segment), yazmaç (register), küme (heap) veya yığıt (stack)), süresini veya ömrünü (programın tüm çalışma süresi veya kodun bazı bloklarının işlenme süreleri) belirler. Depolama sınıfı bildirimin söz dizimi, kaynak kod içerisindeki yeri veya bu etkenlerin her ikisi sayesinde kurulabilir:

```
Depolama-sınıfı tip belirteci
```

mikroC içerisindeki depolama sınıfları şunlardır:

```
auto
register
static
extern
```

Auto depolama sınıfı

`auto` değiştirici kullanımı bir yerel değişkenin yerel süreye sahip olmasını tanımlar. Bu yerel değişkenler için zaten varsayılan durumdur ve değiştirici nadiren kullanılır. `auto`'yu globaller ile kullanamazsınız. Fonksiyonlara da bakınız.

Register depolama sınıfı (Yazmaç depolama sınıfı)

Normalde mikroC, değişkenleri iç mikrodenetleyici belleğine depolar. Bu yüzden `register` değiştiricisinin teknik olarak herhangi özel bir anlamı yoktur. mikroC derleyici basitçe register yerleşimi için olan talepleri ihmal eder.

Static depolama sınıfı

`static` belirteç ile tanımlanan global isim içsel bağlantıya sahiptir. Bu şu anlama gelmektedir: o isim verilen bir dosya için yereldir (local). Daha fazla bilgi için "Bağlantılar" bölümüne bakın.

`static` belirteç ile tanımlanan yerel (local) isim durağan (static) süreye sahiptir. `static` belirtecini yerel değişkenlerin bulundukları fonksiyonun defalarca çağrılmalarında bir önceki değerlerini korumaları için kullanınız. Daha fazla bilgi için "Süreler" bölümüne bakınız.

Extern depolama sınıfı

`extern` ile tanımlanan isim, önceden içsel bağlantı ile bildirilmemiş ise, dışsal bağlantıya sahiptir. Tanımlama, `extern` belirtece sahipse bir tanım olmaz ve ilk değer verilmez. `extern` anahtar kelimesi fonksiyon prototipleri için seçimliktir.

`extern` değiştiricisini bir değişkenin gerçek depo alanının ve ilk değerinin veya bir fonksiyonun gövdesinin bir başka kaynak dosyada tanımlandığını belirtmek için kullanınız. `extern` ile bildirilen fonksiyonlar (fonksiyonu `static` olarak yeniden tanımlamadığınız sürece) bir program içerisindeki tüm kaynak dosyalar tarafından görülebilir.

Daha fazla bilgi için "Bağlantılar" bölümüne bakınız.

Tip Niteleyicileri (Type Qualifiers)

`const` ve `volatile` tip niteleyicileri bildirimlerde seçimliktirler ve gerçekte bildirimi yapılan nesnenin tipine etkileri olmaz.

const Niteleyicisi

`const` niteleyicisi, çalışma süresince değeri değişmeyecek nesne bildirimlerinde uygulanır. `const` niteleyicili bildirimlerde, o bildirim içerisinde nesnelere ilk değerlerini vermelisiniz.

Etkin olarak, mikroC, `const` niteleyicisi ile tanımlanan nesnelere literal veya ön-işlemci sabiti olarak davranır. Derleyici, eğer `const` niteleyicisi ile bildirimi yapılmış bir nesneyi değiştirmeye kalkarsanız, hata mesajı verecektir.

Örnek:

```
const double PI = 3.14159;
```

volatile Niteleyicisi

`volatile` niteleyicisi çalışma süresince değişkenin değerinin programdan bağımsız olarak değişebileceğini belirtir. `volatile` değiştiricisini bir değişkenin arka plandaki bir yordam, bir kesme yordamı veya bir I/O portu ile değiştirilebileceğini bildirmek için kullanınız. Bir nesneyi `volatile` (değişebilir) olarak bildirerek derleyiciyi nesnenin bulunduğu ifadeleri hesaplarken nesnenin değeri ile ilgili varsayımlar yapmaması için uyarmış oluyoruz, çünkü nesnenin değeri herhangi bir anda değişebilir.

typedef Belirteci

`typedef` belirteci bir belirli tip için aynı kavrama karşılık gelen bir isim (eş-isim veya eşanlamlı isim=synonym) tanıtır. `typedef` tanımlamasını kullanarak programlama dili tarafından tanımlanmış veya sizin tarafınızdan tanımlanan tipler için daha kısa ve daha anlamlı isimler oluşturabilirsiniz. `typedef` belirtecini bir fonksiyon tanımlaması içerisinde kullanamazsınız.

`typedef` belirteci bildirim içerisinde ilk sırada bulunur:

typedef <tip-tanımı> eş-isim;

`typedef` anahtar kelimesi *eş-isim*'i *<tip-tanımı>* ile ilişkilendirir. *eş-isim* geçerli bir belirteç olmak zorundadır.

`typedef` belirteci ile başlayan bildirimler verilen tip için bir nesne veya fonksiyon tanıtmaz. Sadece verilen tip için yeni bir isim tanıtır. Şöyle ki `typedef` bildirimi "normal" tanımlama ile aynıdır, fakat nesneler yerine, tiplerin bildirimini yapar. Yaygın olarak benimsenmiş uygulama özel tip tanıtıcılarını büyük harfle başlayacak şekilde isimlendirmektir - Bu C dili için bir gereklilik olmamasına rağmen.

Örnek:

```
// "unsigned long int" icin bir 'es-isim' tanimlayalim:
typedef unsigned long int Mesafe;

// Simdi, "Mesafe" es-ismi bir tip taniticisi olarak
// kullanilabilir
Mesafe i; // unsigned long int olan i degiskenini bildirelim
```

`typedef` bildiriminde herhangi bir bildirimde olduğu gibi, aynı anda birden fazla tip bildirimi yapabilirsiniz. Örnek:

```
typedef int *Pti, Dizi[10];
```

Burada, `Pti` "int cinsine işaretçi" tipi için eş-isim, `Dizi` ise "10 int elemanlı dizi" için eş-isimdir.

asm Bildirimi

C, asm bildirimi sayesinde kaynak kod içerisine Sembolik Makine Dili Komutları (assembly) koyulmasına izin verir. _asm ve __asm tanımlamaları da aynı şekilde mikroC'de kullanılmaktadırlar ve aynı anlamdadırlar. Sembolik Makina Dili komutlarında SFR veya GPR değişkenleri için mutlak adresler için sayısal değerler kullanamazsınız. Ancak onun yerine sembolik isimleri kullanabilirsiniz. (Liste türü çıkış dosyalarında, hem bu isimler hem de adresler görünebilecektir.)

Sembolik Makina Kodu (Assembly) komutlarını `asm` (veya `_asm` veya `__asm`) anahtar sözcüğü ile gruplayabilirsiniz :

```
asm {
   Sembolik Makina Dili Komutları bloğu
}
```

C yorum satırları (hem tek satırlı hemde çok satırlı) gömülü makine dili kodu içerisinde kullanılabilir. Makina dili stili yorumlar olan noktalı virgül ile başlayan yorumların kullanımına izin verilmemektedir.

Eğer belli bir C değişkenini sadece kodunuzun gömülü makina dili kısmında kullanmak istiyorsanız, C kodu içerisinde, en azından ilk değerinin verildiğinden emin olunuz. Aksi taktirde bağlayıcı (linker) bir hata yayınlayacaktır. Bu önceden tanımlı globallere mesela PORTB'ye uygulanmaz.

Aşağıdaki kod derlenemeyecektir, bağlayıcı myvar değişkenini tanıyamayacaktır:

```
unsigned myvar;
void main() {
  asm {
    MOVLW 10  // bu bir testtir
    MOVLW test_main_global_myvar_1
  }
}
```

asm bloğu üzerinde aşağıdaki satırın (veya benzerinin) eklenmesi bağlayıcıya bu değişkenin kullanıldığı bilgisini verir:

```
myvar := 0;
```

Not: Değişkeninize erişim için uygun bankın seçilip seçilmediğini mikroC kontrol etmeyecektir. Sembolik makina kodunda bankları kendiniz seçmelisiniz.

mikroC'nin Eski Versiyonlarından Geçiş

(Eğer mikroC'nin yeni sürümleri ile çalışıyorsanız bu kısmı atlayabilirsiniz.)

asm blokları arasında kullanılan söz dizimi (syntax), versiyon 2'den biraz daha farklıdır. Aradaki farklar:

Örnek olarak değişken isimlendirilmesi:

_myVar, eğer genel değişken ise
FARG_+XX, eğer yerel değişken ise (bu myVar'ın yerel fonksiyon çerçevesindeki asıl pozisyonudur.)
_myVar_L0(+XX), eğer yerel statik değişken ise (+XX daha çok tek-tek baytlara erişim için)

C'deki adları, `asm`'de de aynı kalan tipler sadece sabitlerdir; örneğin:
INTCON, PORTB, WREG, GIE, v.b.

Tek-tek baytlara erişim de farklıdır. Örnek olarak "g_var" gibi bir global değişken'e sahipsek ki, bu uzun tiptir (yani 4 bayt). Ona şu şekilde erişebilirsiniz:

```
MOVF    _g_var+0, 0     ;g_var'ın en alt  baytini W yazmacima (register) koyar.
MOVF    _g_var+1, 0     ; _g_var'ın ikinci bayti; Hi(g_var)'a karsilik gelmektedir.
MOVF    _g_var+2, 0     ;Dah yuksek(g_var)
MOVF    _g_var+3, 0     ; En yuksek(g_var)
...  v.b.
```

Bir nesnenin adres erişimi için sözdizimi farklıdır. Flash ROM'a depolanmış nesneler için:

```
MOVLW   #_g_var         ;adresin ilk bayti
MOVLW   @#_g_var        ;adresin ikinci bayti
MOVLW   @@#_g_var       ;adresin ucuncu bayti
....gibi
```

RAM'e depolanmış nesneler için:

```
MOVLW   CONST1          ;adresin ilk bayti
MOVLW   @CONST1         ;adresin ikinci bayti
.... gibi
```

Başlangıç değeri atama (Initialization)

Bildirim anında, bildirilen nesnenin başlangıç değerini atayabilirsiniz. Başlangıç değeri atamayı belirten bildirim parçasına başlatıcı (initializer) denir.

Global ve static nesneler için başlatıcılar, sabitler ve sabit ifadeler olmalıdır. Otomatik bir nesne için başlatıcı, değişkeninin tipi için atanabilir bir değere indirgenen herhangi bir geçerli ifade olabilir.

Sayısal tipler, seçimlik olarak parantezler içine konabilen tekil ifadeler ile başlatılabilirler. Nesnenin başlangıç değeri bu ifadedir; tip ve dönüşümler için basit atamalar ile aynı kısıtlamalar uygulanır.

Örnek:

```
int i = 1;
char *s = "merhaba";
struct karmasik k = { 0.1, -0.2};
// burada, 'karmasik' bir yapidir.(float, float)
```

Otomatik depolama süreli yapı veya birlikler için, başlatıcı aşağıdakilerden biri olmalıdır:

- bir başlatıcı (yani ön değer verici) listesi,
- uygun birlik ve yapı tipli tek bir ifade. Bu durumda nesnenin başlangıç değeri bu ifadenin değeridir.

Daha fazla bilgi için “Yapılar ve Birlikler”e bakınız.

Aynı zamanda karakter tipinin dizilerini de bir literal karakter dizisi ile başlatabilirsiniz.Bu dizi seçimlik olarak kaşlı parantezler içinde bulunabilir. Karakter dizisi içerisindeki her karakter, Hiçlik sonlandırıcısı (null) dahil, dizi içerisinde ardışık elemanlara başlangıç değerlerini verir. Daha fazla bilgi için “Diziler”e bakınız.

Otomatik Başlangıç Değeri Atama

mikroC, nesneler için otomatik başlangıç değeri sağlamaz. Başlangıç değeri verilmeyen globaller ve statik süreli nesneler bellekten rasgele değerler alacaklardır.

İŞLEVLER (FONKSİYONLAR-FUNCTIONS)

Fonksiyonlar, C programlamanın merkezindedirler. Genellikle birtakım giriş parametrelerine dayanarak bir değer döndüren alt-programcıklar olarak tanımlanırlar. Bir fonksiyonun dönüş değeri ifadeler içinde kullanılabilir. Teknik olarak, fonksiyon çağrısı herhangi diğer bir operatör gibi, değerlendirilir.

C, bir fonksiyonun dönüş değerinden başka sonuçlar üretebilmesine izin verir, bunlar yanal etkilerdir. Sıklıkla fonksiyonun dönüş değeri kullanılmaz ve yanal etkiler göz önüne alınır. Bu tür fonksiyonlar diğer programlama dillerinin (örneğin Pascal dilinin) işlemlerine (prosedürlerine) denktir. C, prosedür ve fonksiyon arasında ayrım yapmaz. Fonksiyon her ikisinin de görevini üstlenebilir.

Her program, programın giriş noktasını belirten ve "main" adında tekil bir dışsal fonksiyona sahip olmalıdır. Fonksiyonlar genellikle standart veya kullanıcının sağladığı başlık dosyalarında veya program dosyalarında prototipler olarak bildirilirler. Fonksiyonlar, varsayılan olarak dışsal bağlantıya (External Linkage) sahiptirler ve programdaki herhangi bir dosyadan erişilebilirler. Bu, fonksiyon bildirimlerinde static depolama sınıf belirtecinin kullanılması ile sınırlanabilir. (Depolama Sınıfları ve Bağlantılara bakınız).

Not: PIC mikrodenetleyicilerinde fonksiyonlarla ilgili sınırlamalar hakkında daha fazla bilgi için "PIC'e Özel" bölümüne bakınız.

Fonksiyon Bildirimi

Fonksiyonlar, kaynak dosyalarınızda bildirilebilir veya ön-derlenmiş kütüphaneleri bağlayarak (link) kullanıma sunulabilirler. Bir fonksiyonun bildirim söz-dizimi:

```
tip fonksiyon_ismi(parametre-tanımlama-listesi);
```

fonksiyon_ismi geçerli bir tanıtıcı olmalıdır. Bu isim fonksiyonu çağırmak için kullanılır. Daha fazla bilgi için "Fonksiyon Çağrımları" bölümüne bakınız. *tip* fonksiyon sonucunun tipini gösterir, ve herhangi standart veya kullanıcı tanımlı tip olabilir. Değer döndürmeyen fonksiyonlar için `void` tipini kullanmalısınız. *tip* global fonksiyon bildirimlerinde gözardı edilebilir ve fonksiyon varsayılan (default) olarak `int` tipinde varsayar.

tip aynı zamanda bir işaretçi de olabilir. Örnek olarak, `float*` şu anlama gelir; fonksiyon sonucu, bir `float`'a işaret eden bir işaretçidir. Soysal işaretçi olan `void*` 'a daima izin verilir. Fonksiyon dizi veya başka bir fonksiyon döndüremez.

Parantezler içerisindeki *parametre-tanımalama-listesi* fonksiyonun aldığı formal bağımsız değişkenlerin listesidir. Bu bildiriciler her fonksiyon parametresinin tipini belirtirler. Derleyici, bu bilgileri fonksiyon çağırımlarının doğruluğunu denetlemek için kullanır. Eğer liste boş ise, fonksiyon herhangi bir bağımsız değişken almaz. Aynı zamanda eğer liste `void` ise fonksiyon herhangi bir bağımsız değişken almaz; dikkat edin ki, bu `void`'in bir bağımsız değişkenin tipi olarak kullanıldığı tek durumdur.

Değişken bildiriminden farklı olarak, liste içerisindeki her bağımsız değişken özellikle kendi tip belirleyicisine ve olası bir `const` veya `volatile` niteleyiciye ihtiyaç duyar.

Fonksiyon Prototipleri

Verilen bir fonksiyon bir programda ancak bir kez tanımlanabilir, fakat birkaç kez bildirilebilir (yeter ki bildirimler birbirleri ile uyumlu olsun). Eğer bir fonksiyonun tanımlama yapmayan bir bildirimini yazarsanız (yani fonksiyon gövdesi yok ise) formal bağımsız değişkenlerin adlarını belirtmek zorunda değilsiniz. Bu tür bir bildirim, genel olarak *fonksiyon prototipi* olarak bilinir ve size; bağımsız değişken sayıları, tip denetleme ve tip dönüştürme üzerinde daha iyi bir kontrol sağlar.

Fonksiyon prototipi içerisindeki isim, prototip ile sınırlı kendi kapsamına sahiptir. Bu aynı fonksiyonun farklı bildirimlerinde farklı parametre isimleri kullanılabilmesini sağlar.

```
/* Aşağıda aynı fonksiyonun iki ayrı prototipi verilmiştir: */

int test(const char*)    // test fonksiyonunu bildirir
int test(const char* p)  // aynı test fonksiyonunu bildirir
```

Fonksiyon prototipleri kodun dökümanını hazırlamada büyük yardımcıdır. Örnek olarak: `Cf_Init` fonksiyonu iki parametre alır: `ctrlport` ve `dataport`. Peki hangisi önce hangisi sonra gelir? Fonksiyon prototipi,

```
void Cf_Init(char *ctrlport, char *dataport);
```

bunu açıklığa kavuşturur. Eğer bir başlık dosyası fonksiyon prototipilerini içeriyor ise, bu dosyayı programları yazarken ve fonksiyonları çağırırken ihtiyacınız olan bilgileri elde etmek için kullanabilirsiniz. Eğer bir prototip parametresi içerisine bir tanıtıcı (ismini) ilave ederseniz bu sadece o parametreyi içeren hata mesajları için kullanılır. Başka herhangi bir etkiye sahip değildir.

Fonksiyon Tanımlama

Fonksiyon tanımlama, fonksiyon bildirimi ve bir fonksiyon gövdesinden oluşur. Fonksiyon gövdesi teknik olarak bir bloktur çünkü Kaşlı Parantezler {} ile sınırlandırılmış bir dizi yerel tanım ve deyimden oluşur. Fonksiyon gövdesi içinde bildirilen tüm değişkenler fonksiyon için yereldir, yani fonksiyon kapsamına (scope) sahiptirler.

Fonksiyonun kendisi sadece dosya kapsamı içerisinde tanımlanabilir. Bu, şu anlama gelmektedir: fonksiyon tanımlamaları iç-içe geçmiş olamaz.

Fonksiyon sonucunu döndürmek için `return` deyimi kullanılır. `void` tip olan fonksiyonlardaki `return` deyimi bir parametreye sahip olamaz. Aslında, `return` deyimi, fonksiyon gövdesi içerisinde son deyim ise, bu deyimi tamamen ihmal bile edebilirsiniz.

Aşağıda örnek bir fonksiyon tanımı yer almaktadır:

```
/*max fonksiyonu iki bagimsiz degiskeninden buyugunu dondurur.*/
int max(int x, int y) {
  return (x>=y) ? x : y;
}
```

Aşağıda, dönüş değerinden ziyade yanal etkilere (side-effects) dayanan örnek fonksiyon verilmiştir:

```
/* fonksiyon (x,y) duzlemsel koordinatlarini (r,fi) kutupsal koor-
dinatlarina donusturuyor: */

#include <math.h>

void polar(double x, double y, double *r, double *fi) {
  *r = sqrt(x * x + y * y);
  *fi = (x == 0 && y == 0) ? 0 : atan2(y, x);
  return; /* bu satır ihmal edilebilir. */
}
```

Fonksiyonların Yineli-girişliliği (Reenterancy)

Limitli olarak fonksiyonlar için (daha çıkmadan) yine-giriş'e izin verilir. Kendi fonksiyon çerçevelerine (frame) sahip olmayan fonksiyonlar yani bağımsız değişkenleri ve yerel değişkenleri olmayan fonksiyonlar, hem kesmeden (interrupt) hem de "ana" thread'tan çağrılabilirler. Bağımsız girdi değişkenlerine sahip (yani argumanları olan) ve/veya yerel değişkenleri olan fonksiyonlar az önce bahsettiğimiz program iş parçacıklarından (thread'lar) sadece birinden çağrılabilirler. *Dolaylı Fonksiyon Çağrımlarına* bakınız.

Fonksiyon Çağrıları (Function Calls)

Bir fonksiyon, kendi biçimsel parametreleri gibi aynı sırada yerleştirilmiş, bire bir uygun, gerçek bağımsız değişkenler ile çağrılabilir. Fonksiyon çağırma operatörü olan ()'yı şu şekilde kullanınız:

```
fonksiyon_ismi(ifade_1, ... , ifade_n)
```

Fonksiyon çağrımı içerisindeki her ifade bir fiili bağımsız değişkendir (argument). Gerçek bağımsız değişkenlerin sayı ve tipleri tüm biçimsel fonksiyon parametreleri ile uyuşmalıdır. Eğer tip uyuşmuyorsa, içsel tip dönüştürme kuralları uygulanır. Gerçek bağımsız değişkenler herhangi bir karmaşıklıkta olabilirler. Fakat, siz onların göz önüne alınma sırasına güvenmeyiniz, çünkü böyle bir sıra belirtilmemiştir.

Fonksiyon çağrısı oluştuğunda, tüm biçimsel parametreler, fiili bağımsız değişkenlerin değerleri ile başlatılmış yerel nesneler gibi oluşturulur. Bir foksiyondan dönüş oluştuğunda, geçici bir nesne, az önce çağrının olduğu yerde oluşturulur ve `return` deyiminindeki ifadenin sonucu ile ilk değer olarak atanır. Bu şu anlama gelmektedir; karmaşık bir ifadede "operand" olarak kullanılan fonksiyon çağrısı, fonksiyonun sonucu olarak göz önüne alınır.

Eğer fonksiyon sonuçsuz bir fonksiyon ise (`void` tipi) veya sonuca ihtiyacınız yoksa, fonksiyon çağrısını kendi başına bir ifade olarak yazabilirsiniz.

C'de, sayısal parametreler, fonksiyona daima değer geçişi yöntemiyle geçerler. Bir fonksiyon kendi parametrelerinin değerlerini değiştirebilir fakat bu, çağrıyı yapan yordam içindeki gerçek bağımsız değişkenleri etkilemeyecektir. Sayısal nesneyi, adres ile, bir biçimsel parametreyi bir işaretçi olarak bildirerek geçirebilirsiniz. Sonrasında, işaret edilen nesneye erişim için 'dolaylı erişim operatörü' olan yıldız işaretini (*) kullanınız.

Bağımsız Değişken Dönüşümleri

Bir fonksiyon prototipi önceden bildirilmediğinde, mikroC, Standart Dönüşümlerde tanımlanan tümleşik genişletme kurallarına uygun olarak fonksiyon çağrısındaki tümleşik bağımsız değişkenleri dönüştürür. Bir fonksiyon prototipi kapsam içerisinde olduğunda, mikroC, verilen bağımsız değişkenlerin değerlerini bildirimi yapılan parametrelerin tiplerine dönüştürür.

Eğer bir prototip varsa, bağımsız değişkenlerin sayısı uyuşmalıdır. Tipler ise bir atama işleminin kurallara uygun dönüştürümü sağlayabileceği kadar uyuşmalıdırlar. Tabi ki her zaman bir bağımsız değişkeni fonksiyon prototipinin kabul edebileceği bir tipe dönüştürmek için açık dönüştürme işlemini (cast) kullanabilirsiniz.

Not: Eğer fonksiyon prototipiniz gerçek fonksiyon tanımı ile uyuşmuyorsa, mikroC bu hatayı sadece ve sadece fonksiyon tanımı ve fonksiyon prototipi aynı derleme ünitesi içinde olurlarsa tesbit edebilir. Eğer yordamlarınızdan bir kütüphane üretir ve ilgili başlık dosyasını da hazırlarsanız, kütüphanenizi derlerken başlık dosyasını da içermeyi (include etmeyi) önemle düşününüz, bu sayede prototipleriniz ve gerçek tanımlarınız arasındaki çelişkileri kolayca tesbit edebilirsiniz.

Derleyici, daima bağımsız değişkenleri uygun tipe dönüştürmeye çalışacaktır. Varsayalım ki aşağıdaki koda sahipsiniz:

```
int limit = 32;
char ch = 'A';
long res;

extern long func(long par1, long par2); // prototip

main() {
  //...
  res = func(limit, ch);               // fonksiyon cagrisi
}
```

`func` için fonksiyon prototipine sahip olduğu için, bu program, standart atama kurallarını kullanarak, `limit` ve `ch` yi, `func` çağrımı için yığıta (stack) yerleştirmeden önce, `long`'a dönüştürür.

Fonksiyon prototipi olmasaydı, `limit` ve `ch` yığıt üzerine sırasıyla bir tamsayı ve bir karakter şeklinde yerleşecektiler; bu durumda `func`'una geçen yığıt bölümü; fonksiyonun beklediği boyut ve içerik ile uyuşmayacak, problemler ortaya çıkacaktı.

Üç Nokta Operatörü : ('...') (Ellipsis Operator)

Üç nokta (ellipsis) operatörü yani simge olarak ('...'), arada beyaz boşluk (white-space) bulunmayan üç adet ardışık noktadan oluşur. Üç noktayı, değişebilen sayıda bağımsız değişken olduğunu veya tipleri değişken olan bağımsız değişkenleri göstermek için fonksiyon prototipinin biçimsel bağımsız değişkenler listesinde kullanabilirsiniz. Örnek olarak:

```
void func (int n, char ch, ...);
```

Bu bildirim şu anlama gelmektedir: Fonksiyon öyle tanımlanmalıdır ki en azından iki tane bağımsız değişkeni olsun, bunlardan ilki *int* ve diğeri ise *char* tipinde olsun ve aynı zamanda da değişken sayıda bağımsız değişkeni olsun.

Örnek:

```
#include <stdarg.h>


int addvararg(char a1,...){
va_list ap;
char temp;
va_start(ap,a1);

while( temp = va_arg(ap,char))
  a1 += temp;
return a1;
}


int res;
void main() {


  res = addvararg(1,2,3,4,5,0);

  res = addvararg(1,2,3,4,5,6,7,8,9,10,0);

} //~!
```

OPERATÖRLER

Operatörler, bir ifadenin değişken ve diğer nesnelerine uygulandığında bazı hesaplamaları tetikleyen dizgecikler veya sembollerdir.

mikroC aşağıdaki operatörleri tanımlar:

- Aritmetik Operatörler
- Atama (Assignment) Operatörleri
- Bit-işlem Operatörleri
- Mantıksal (Logical) Operatörler
- Başvuru/Dolaylı Erişim Operatörleri (İşaretçi Aritmetiğine bakın)
- İlişkisel (Relational) Operatörler
- Yapı Üyesi Seçicileri (Yapı Üyesi Erişimine bakın)

- Virgül Operatörü ‘`,`’ (Virgül İfadelerine bakın)
- Şartlı Operatörler ‘`?` `:`

- Dizi indeks operatörü ‘`[]`‘ (Dizilere bakın)
- Fonksiyon çağrı operatörü ‘`()`‘ (Fonksiyon Çağrılarına bakın)

- `sizeof` Operatörü (Ne büyüklükte operatörü)

- Ön-işlemci Operatörleri `#` and `##` (Ön-işlemci Operatörlerine bakın)

Operatörlerin Öncelik ve Matematiksel Birleşme Özellikleri

mikroC’de 15 tane öncelik kategorisi mevcuttur, bazı kategorilerde sadece bir operatör vardır. Aynı kategorideki operatörler eşit önceliğe sahiptirler.

Bir sonraki sayfadaki tablo, tüm mikroC operatörlerini bulundurmaktadır.

Tabloda operatörlerin bazıları iki kez görünse de ilk görünüş tekli operatörü, ikinci görünüş ise ikili operatörü göstermektedir. Her kategori bir birleşme özelliği kuralına sahiptir: soldan sağa veya sağdan sola. Parantezler olmadığı zaman, bu kurallar eşit öncelikteki operatörlerin bulunduğu ifadelerin gruplanmasını sağlar.

Öncelik	Operand (İşlenen)	Operatör (İşleyen)	Birleşirlik
15	2	`()  []  .  ->`	soldan sağa
14	1	`!  ~  ++  --  +  -  *  &  (type)  sizeof`	sağdan sola
13	2	`*  /  %`	soldan sağa
12	2	`+  -`	soldan sağa
11	2	`<<  >>`	soldan sağa
10	2	`<  <=  >  >=`	soldan sağa
9	2	`==  !=`	soldan sağa
8	2	`&`	soldan sağa
7	2	`^`	soldan sağa
6	2	`|`	soldan sağa
5	2	`&&`	soldan sağa
4	2	`||`	soldan sağa
3	3	`?:`	soldan sağa
2	2	`=  *=  /=  %=  +=  -=  &=  ^=  |=  <<=  >>=`	sağdan sola
1	2	`,`	soldan sağa

Aritmetik Operatörler

Aritmetik operatörler, matematiksel hesaplamaları gerçekleştirmek için kullanılırlar. Sayısal işlenenler (operand) üzerinde çalışırlar ve sayısal sonuçlar üretirler. `char` tipi teknik olarak küçük tamsayıları temsil ettiğinden aritmetik işlemlerde kullanılabilir.

Tüm aritmetik operatörler soldan sağa doğru birleştirilirler.

Operator	Operasyon (İşlem)	Öncelik
+	toplama	12
−	çıkarma	12
*	çarpma	13
/	bölme	13
%	tamsayı bölümünün kalanını döndürür (kayan noktalılarla kullanılmaz)	13
+ (tekli)	tekli artı, işlenene (operanda) etki etmez	14
− (tekli)	tekli eksi, işlenenin (operandın) işaretini değiştirir.	14
++	işlenenin (operandın) değerini bir arttırır.	14
−−	işlenenin (operandın) değerini bir azaltır	14

Not: `*` `operatörü` içeriğe duyarlıdır ve ayrıca işaretçi referans operatörünü de temsil edebilir. Daha fazla bilgi için “İşaretçilere” bakın.

İkili Aritmetik Operatörler

İki tamsayının bölümünde, kalan basitçe atılır ve sonuç tamsayı olarak döner,

```
/* örnek: */
7 / 4;          // esittir 1
7 * 3 / 4;      // esittir 5

/* fakat: */
7. * 3./ 4.;  // esittir 5.25 (eger float'larla calisirsak)
```

Kalan operatörü olan '%' sadece tamsayılarla çalışır. Sonucun işareti ilk işlenenin (operandın) işaretidir:

```
/* örnek: */
2 % 3;        // esittir 0
7 % 3;        // esittir 1
-7 % 3;       // esittir -1
```

Aritmetik operatörlerini karakterleri işlemek için kullanabiliriz:

```
'A' + 32;           // = 'a' (ASCII sadece)
'G' - 'A' + 'a';    // = 'g' (hem ASCII hem de EBCDIC)
```

Tekli Aritmetik Operatörler

++ ve -- tekli operatörleri C'de ön-ek (++k, --k) veya son-ek (k++, k--) olabilen yegane operatörlerdir.

++ ve -- tekli operatörleri (önce-arttır ve önce-eksilt operatörleri) ön-ek olarak kullanıldıklarında bulundukları ifadenin değerlendirilmesinden önce işleneni bir arttırır veya azaltırlar. Son-ek olarak kullanıldıklarında bulundukları ifadenin değerlendirilmesinden sonra eklendikleri operandın değerini bir arttırır veya bir azaltırlar.

Örnek:

```
int j = 5; j = ++k;
/* k = k + 1, j = k, bize su sonucu verir: j = 6, k = 6 */

int j = 5; j = k++;
/* j = k, k = k + 1, bize su sonucu verir: j = 5, k = 6 */
```

İlişkisel Operatörler (Relational Operators)

İlişkisel operatörleri ifadelerin eşit veya eşit olmadıklarını test etmek için kullanınız. Eğer ifade "doğru" ('true') değeri alırsa, 1 döner; aksi halde 0 döner.

Tüm ilişkisel operatörler soldan sağa birleşirler.

ilişkisel operatörler:

Operatör (İşleyen)	İşlem	Öncelik
==	eşit	9
!=	eşit değil	9
>	büyük	10
<	küçük	10
>=	büyük veya eşit	10
<=	küçük veya eşit	10

İfadelerde İlişkisel Operatörler

Aritmetik ve ilişkisel operatörlerin önceliği; karmaşık ifadelerde, ifadenin parantezsiz olarak da beklenen anlamı vermesine imkan verecek şekilde düzenlenmiştir:

```
a + 5 >= c - 1.0 / e    // i.e. (a + 5) >= (c - (1.0 / e))
```

Unutmayın ki ilişkisel operatörler daima ya 0 ya da 1 döndürürler. Aşağıdaki örneklerde olduğu gibi :

```
8 == 13 > 5       // donen deger 0: 8==(13>5), 8==1, 0
14 > 5 < 3        // donen deger 1: (14>5)<3, 1<3, 1
a < b < 5         // donen deger 1: (a<b)<5, (0 or 1)<5, 1
```

Bit-İşlem (Bitwise) Operatörleri

Bit-işlem operatörünü sayısal bir işlenenin bitlerini değiştirebilmek amacıyla kullanırız.
Bit-işlem operatörleri soldan sağa doğru birleşir. Sadece kural dışı olarak, tümleyici bit-işlem operatörü '~' soldan sağa doğru birleşmez.

bit işlem operatörleri:

Operatör	İşlem	Öncelik
&	bit-işlem VE; eğer her iki bit de 1 ise, 1 döndürür, diğer durumlarda 0 döndürür	9
|	bit-işlem VEYA; bitlerden biri veya her ikisi 1 ise 1 döndürür, diğer durumda 0 döndürür.	9
^	Dışlayan VEYA (Exclusive XOR); eğer bitler tümleyici ise 1, diğer durumda 0 döndürür	10
~	Tümleyici bit-işlem operatörü; Her bitin evriğini alır (teklidir; tek işlenene etki eder)	10
>>	Bitleri sola kaydır; Bitleri sola taşır, Bilgi için aşağıya bakınız	10
<<	Bitleri sağa kaydır; Bitleri sağa taşır, Bilgi için aşağıya bakınız	10

Not: & operatörü aynı zamanda işaretçi başvuru operatörü de olabilir. Daha fazla bilgi için 'İşaretçiler' bölümüne bakın.

&, |, ve ^ bit-işlem operatörleri operandlarının karşılıklı bit çiftleri üzerinde mantıksal işlem gerçekleştirirler. Örnek:

```
0x1234 & 0x5678;        /* sonucu esittir  0x1230 */

/* cunku detaylara inersek ..

0x1234      : 0001 0010 0011 0100
0x5678      : 0101 0110 0111 1000
--------------------------------
  &   : 0001 0010 0011 0000

.. yani sonuc, 0x1230'dur */
```

```
/* Benzer sekilde: */

0x1234 | 0x5678;    /* esittir 0x567C */
0x1234 ^ 0x5678;    /* esittir 0x444C */
~ 0x1234;           /* esittir 0xEDCB */
```

Bit-işlem Kaydırma Operatörleri

İkili (binary) operatörlerden ‘<< ‘ ve ‘>> ‘ sol işlenenin (operand) bitlerini, sağ işlenen tarafından belirlenen kadar, sola veya sağa kaydırırlar. Sağ işlenen pozitif olmalıdır.

Sola kaydırma ile (<<), en soldaki bitler atılır ve yeni bitler sağ taraftan 0 olarak atanır. Dolayısıyla işaretsiz bir işleneni sola (n) tane kaydırmak; kaydırma işleminde düşen bitler sıfır ise, o işleneni (2^n) ile çarpmak demektir. Bu işlem, kaydırma işleminde düşen bitler işaret biti ise, işaretli işlenenler için de geçerlidir.

```
000001 <<  5;           /* = 000040 */
0x3801 <<  4;           /* = 0x8010, tasma var! */
```

Sağa kaydırma ile (>>) , en sağdaki bitler atılır ve soldaki boşalan bitlerin yerlerine, eğer işlenen işaretsiz ise sıfırlar, işaretli ise işaret biti atanır. Sağa (n) kez kaydırmak, işleneni (2^n) ile bölmek demektir.

```
0xFF56  >>  4;      /* = 0xFFF5 */
0xFF56u >>  4;      /* = 0x0FF5 */
```

Bit-işlem ve Mantık operatörlerinin karşılaştırmalı incelemesi

Bit-işlem ve mantık operatörünün çalışma ilkelerinin farklı olduğunu farkediniz. Örnek:

```
0222222 &  0555555;       /* esittir 000000 */
0222222 && 0555555;       /* esittir 1 */

~ 0x1234;                 /* esittir 0xEDCB */
! 0x1234;                 /* esittir 0 */
```

Mantıksal Operatörler

Mantıksal operatörlerin işlenenleri (operandları) *doğru* veya *yalnış* olarak; başka bir değişle *sıfır* veya *sıfır olmayan* şeklinde göz önüne alınırlar. Mantıksal operatörler daima 1 veya 0 döndürürler. Mantıksal ifade içerisindeki işlenenler sayısal olmalıdırlar.

`&&` ve `||` mantıksal operatörleri soldan sağa birleşirler. Mantıksal olumsuzlama (değil) operatörü olan `!` ise sağdan sola doğru birleşir.

Operatör	İşlem	Öncelik
&&	Mantıksal AND	5
||	Mantıksal OR	4
!	Mantıksal olumsuz (değil)	14

Mantıksal Operatörler:

Mantıksal, ilişkisel ve aritmetik operatörlerin işlem öncelikleri seçilirken, mümkün olduğunca karmaşık ifadelerin parantezsiz kullanımlarının da beklenen değeri vermesi prensibine dikkat edilmiştir.

```
c >= '0' && c <= '9'; // su demektir: (c>='0') && (c<='9')
a + 1 == b || ! f(x;) // su demektir: ((a+1)== b) || (!(f(x)))
```

Mantıksal AND (`&&`) için eğer her iki yanındaki ifade de sıfır olmayan bir değer ile sonuçlanıyorsa 1 döner, aksi taktirde 0 döner. Eğer ilk ifade *yalnış* yani *sıfır* olarak değerlendirilirse, ikinci ifade değerlendirilmez. Örnek olarak:

```
a > b && c < d;      // su demektir: (a>b) && (c<d)
// Eger (a>b) yanlis ise (0), (c<d) hesaplanmayacaktir.
```

Mantıksal OR (`||`) için eğer her iki yanındaki ifadelerden biri sıfırla sonuçlanmıyorsa 1 döner. Aksi taktirde 0 döner. Eğer ilk ifade *doğru* yani *bir* olarak değerlendirilirse ikinci ifade değerlendirilmez. Örnek olarak:

```
a && b || c && d;    // su demektir: (a && b) || (c && d)
// Eger (a&&b) doğru ise(1), (c&&d) hesahesaplanmayacaktir.
```

Mantıksal İfadeler ve Yan Etkiler

Karmaşık ifadeler ile ilgili genel kural şudur: Nihai sonuç belli olur olmaz, izleyen ifadelerin hesaplanması durdurulur. Örnek verirsek, aşağıdaki ifadeyi göz önüne aldığımızda:

```
a && b && c
```

Burada `a` yalnış (0) ise, daha sonrasında b ve c göz önüne alınmayacaktır. Bu durum eğer b ve c ifade iseler çok önemlidir, çünkü artık hesaplanmayacakları için etkleri de kaybolacaktır!

Mantıksal ve Bit-işlem

Bit-işlem ve mantıksal operatörlerinin çalışma kuralları arasındaki farklılığın farkında olunuz. Örnek olarak:

```
0222222 &  0555555        /* esittir 000000 oysa */
0222222 && 0555555        /* esittir 1 */

~ 0x1234                  /* esittir 0xEDCB oysa */
! 0x1234                  /* esittir 0 */
```

Koşul Operatörü (? :)

Koşul operatörü olan '? :' C'deki tek üçlü operatördür. Koşul operatörünün sözdizimi:

```
ifade1 ? ifade2 : ifade3
```

ifade1 en önce değerlendirilir. Eğer onun değeri *doğru* ise *ifade2* değerlendirilir ve *ifade3* yok sayılır (ihmal edilir). Ancak eğer değerlendirmede *ifade1* *yalnış* olarak sonuç verirse bu kez *ifade3* değerlendirilir ve *ifade2* yok sayılır. Tüm ifade grubunun sonucu hangisinin değerlendirildiğine bağlı olarak *ifade2* veya *ifade3*'ün sonucu olacaktır. Eğer bu iki ifadeden sadece bir tanesinin değerlendirilmeye alınması gerçeği, eğer ifadelerin beklediğiniz başka etkileri olacağını düşündüyseniz, çok önemlidir!

Koşul operatörü sağdan sola birleşir.

Aşağıda yararlı bir çift örnek bulabilirsiniz:

```
/*  max(a, b)'i bul: */
max = (a > b) ? a : b;

/* Kucuk harfi buyuk harfe cevir: */
/* (aslinda burada parantezlerin hicbiri gerekli degil) */
c = (c >= 'a' && c <= 'z') ? (c - 32) : c;
```

Koşul Operatörü Kuralları

ifade1 bir sayısal ifade olmalıdır. *ifade2* ve *ifade3* de aşağıdaki kurallardan birine uymalıdırlar:

1. Her ikisi birden aritmetik tip olabilir; *ifade2* ve *ifade3* alışılmış aritmetik çevirmelere tabiidir, ve bu tip genel ifade sonucunu belirler.
2. Her ikisi birden uyumlu yapı veya birlik tipleri olabilir; Genel ifade sonuç tipi *ifade2* ve *ifade3*'ün yapı ve birlik tipindedir.
3. Her ikisi birden `void` tipinde olabilir; Genel ifade sonucu `void` tipindedir.

4. Her ikisi de uyumlu tiplerin nitelikli veya niteliksiz türlerine işaretçi olabilirler; Bu durumda genel ifade sonucu her iki işaretçi tarafından gösterilen tiplerin tüm niteleyicileri tarafından nitelenen türü gösteren bir işaretçidir.

5. Operandlardan biri işaretçi ve diğeri bir hiçlik (null) işaretçi sabiti olabilir. Bu durumda genel ifade sonucu her iki işaretçi tarafından gösterilen tiplerin tüm niteleyicileri tarafından nitelenen türü gösteren bir işaretçidir.

6. Operandlardan biri bir nesneyi veya tamamlanmamış bir tipi gösteren bir işaretçi olabilir ve diğeri de nitelikli veya niteliksiz `void` türüne bir işaretçi olabilir. Sonuç `void` olmayan tipi işaret eden bir işaretçi tipindedir.

Atama Operatörleri

Diğer birçok programlama dillerinden farklı olarak, C veri atama işlemini bir komuttan ziyade bir işlem (bir operatör tarafından temsil edilen) olarak işler.

Basit Atama Operatörü

Alışıldık değer atamasında, basit atama operatörü olan eşittir'i (=) kullanırız

```
ifade1 = ifade2
```

ifade1 , *ifade2* nin değerini atadığımız bir nesnedir (hafıza bölgesi). *ifade1* işleneni (operandı) bir ldeğer olmak zorundadır, *ifade2* herhangi bir ifade olabilir. Atama ifadesi tek başına bir ldeğeri değildir.

Eğer *ifade1* ve *ifade2* farklı tipten iseler, *ifade2*'nin sonucu gerektiğinde *ifade1* 'in tipine dönüştürülecektir.

Birleşik Atama Operatörleri

C, birleşik atama operatörleri sayesinde karmaşık atamalar oluşturmanıza izin verir. Birleşik atama operatörlerinin sözdizimi:

```
ifade1 op= ifade2
```

burada *op* şu ikili operatörlerden biri olabilir: `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, veya `>>`.

Böylelikle, 10 farklı birleşik atama operatörüne sahip olmuş oluyoruz: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, ve >>=. Bunların hepsi sağdan sola birleşirler. Boşluk ile birbirinden ayrılan operatörler hata oluştururlar (örnek: + =)

Birleşik atama aşağıdaki deyim ile aynı etkiye sahiptir

```
ifade1 = ifade1 op ifade2
```

ancak tek farklılık, bir ldeğer olan *ifade1* sadece bir kez değerlendirilir. Örnek:

```
ifade1 += ifade2
```

aşağıdaki deyim ile aynıdır.

```
ifade1 = ifade1 + ifade2
```

Atama Kuralları

Hem basit hem de birleşik atama için, *ifade1* ve *ifade2* işlenenleri (operandları) aşağıdaki kurallardan birine uymak zorundadırlar:

1. `ifade1` nitelikli veya niteliksiz bir aritmetik tiptir ve *ifade2* bir aritmetik tiptir.

2. `ifade1` , `ifade2`'nin tipiyle uyumlu bir yapı veya birliğin nitelikli veya niteliksiz bir versiyonuna sahiptir.

3. `ifade1` ve `ifade2` uyumlu tiplerin nitelikli veya niteliksiz versiyonlarına işaretçidirler. Sol tarafta işaret edilen tip sağ tarafta işaret edilen tipin tüm niteleyicilerine sahiptir.

4. Ya `ifade1` ya da `ifade2` bir nesnenin veya tam olmayan bir tipin bir işaretçi sidir. Diğeri ise void'in nitelikli veya niteliksiz versiyonunun bir işaretçisidir. Sol tarafta işaret edilen tip sağ tarafta işaret edilen tipin tüm niteleyicilerine sahip tir.

5. `ifade1` bir işaretçidir ve `ifade2` bir hiçlik (null) işaretçi sabitidir.

Sizeof Operatörü

Bir önek olan tekli (tek işlenenli) `sizeof` operatörü, işleneni tarafından bellek bölümünün ne kadar kullanıldığını bayt olarak veren bir tamsayı sabiti döndürür. (bazı istisnai durumlar dışında bu büyüklük işlenenin tipi sayesinde belirlenir).

`sizeof` operatörü, işlenen olarak ya bir tip niteleyici ya da bir tekli bir ifade alabilir. `sizeof`'u, fonksiyon tipinin, eksik tiplerin ve böyle tiplerin parantezli isimlerinin ifadeleri ile kullanamazsınız. Ayrıca `sizeof`'u, bir bit alanı nesnesini oluşturan bir ldeğeri ile kullanamazsınız.

İfadelere Uygulanan Sizeof

Eğer bir ifadeye uygulanırsa, operandın boyutu ifade değerlendirilmeden hesaplanır (Yani ifadenin değerlendirilmesinden dolayı bir yan etki olmayacaktır). İşlemin sonucu, ifadenin (eğer değerlendirilseydi beklenen) sonucunun tipinin boyutu olacaktır.

Tipe Uygulanan Sizeof

Eğer bir tip tanıtıcısına uygulanırsa, `sizeof,` belirtilen tipin boyutunu döndürür. Tip boyutu için birim bir byte'a eşit olan `sizeof(char)`'dır. `sizeof(char)` işlemi, karakterin işaretli veya işaretsiz olmasını gözetmeden 1 sonucunu verir.

```
sizeof(char)              /* 1 dondurur; mikroC icin*/
sizeof(int)               /* 2 dondurur; mikroC icin */
sizeof(unsigned long)     /* 4 dondurur; mikroC icin */
```

Operand, fonksiyon parametresi olmayan bir dizi olduğunda, sonuç dizi içerisindeki toplam bayt sayısıdır (başka bir deyişle, bir dizi ismi bir işaretçi tipine dönüştürülmez ve dizinin boyutu döndürülür.):

```
int i, j, a[10];
//...
j = sizeof(a[1]);         /* j = sizeof(int) = 2 */
i = sizeof(a);            /* i = 10*sizeof(int) = 20 */
```

Eğer operand, fonksiyon parametresi olan bir dizi veya fonksiyon tipinde ise `sizeof` işaretçinin boyutunu verir. Yapılara veya birliklere uygulandığında, `sizeof` operatörü, dolgu baytlarını da hesaba katarak, toplam baytların sayısını verir. `sizeof` operatörü bir fonksiyona uygulanamaz.

İFADELER (EXPRESSIONS)

Bir ifade, bir hesaplamayı belirten işlemler (operatörler), işlenenler (operandlar) ve noktalama işaretleri dizisidir. Biçimsel olarak, ifadeler özyineli (recursive) olarak tanımlanırlar: alt ifadeler biçimsel bir limit olmadan iç-içe (nested) koyulabilirler. Ancak, derleyici çok karmaşık bir ifadeyi derleyemezse, belleğin aşıldığını belirten bir hata raporunu üretecektir.

ANSI C'de, birincil ifadeler şunlardır: sabitler (aynı zamanda literaller olarak da kullanılmaktadırlar), tanıtıcılar ve özyineli (recursive) olarak tanımlı ifadeler.

İfadeler, parantezlerin varlığına, kullanılan operatörlere ve işlenenlerin veri tipine bağlı olarak dönüşüm, gruplama, birleşim ve öncelik kurallarına tabi tutulurlar. Operatörlerin öncelik ve birleşimi için 'Operatör Önceliği ve Birleşirlik' bölümüne bakınız. Operatörlerin ve alt ifadelerin gruplama şekilleri, bunların mikroC'deki gerçek değerlendirme sırasını göstermeyebilir.

İfadeler, bir ldeğer veya bir rdeğer üretebilir veya hiçbir değer üretmeyebilir. İfadeler bir değer üretip üretmediklerine bağlı olmaksızın yanal etkilere sahip olabilirler.

Virgül İfadeleri

C'nin özelliklerinden biri, sıralamalar veya sözde virgül ifadelerini oluşturmak için, virgülü bir sıralama operatörüymüş gibi kullanmanıza olanak sağlamasıdır. Virgül ifadeleri virgülle sınırlandırılmış ifadeler listesidir. Biçimsel olarak bir tek ifade gibi ele alındığından, bir ifadenin beklenildiği her yerde kullanılabilir:

```
ifade_1, ifade_2;
```

Yukarıdaki ana ifade, soldan sağa her ifade için değerlendirilir. Tüm ifadenin değeri ve tipi *ifade_2'*nin tip ve değeri olur. ifade_1'in değeri (yani sonucu) gözardı edilir.

İkili operatör olan virgül (,) en düşük önceliğe sahiptir ve soldan sağa doğru birleşir. Bu nedenle `a, b, c` şu şekilde birleşir : `(a, b), c`. Bu bize herhangi sayıda ifade kullanarak sıralamalar yazmamız olanağını sağlar:

```
ifade_1, ifade_2, ... ifade_n;
```

Yukarıdaki ana ifade her bireysel ifadenin soldan sağa tek tek hesaplanması ve ana ifadenin sonucunun ve tipinin *ifade_n* 'in tipi ve değeri olması ile sonuçlanır. Diğer ifadelerin sonuçları gözardı edilir, fakat onların değerleri de hesaplandığı için yanal etkiler oluşabilir.

İfade sonucuna ve yanal etkilere örnek verecek olursak:

```
sonuc = (a = 5, b /= 2, c++);
/* c degiskeninin onceden artisli degeri sonuc olarak doner,
fakat ayni zamanda a'ya ilk deger verilir, b 2'ye bolunur ve
ve c arttirilir*/

sonuç = (x = 10, y = x + 3, x--, z -= x * 3 - --y);
/* z degiskeninin hesaplanmis degeri sonuc olarak doner
   ve ayni zamanda x ve y de hesaplanir */
```

Not: Virgül operatörünü (Sıralama operatörü), bir fonksiyon bağımsız değişken listesindeki veya başlatıcı listelerindeki elemanları birbirinden ayırma işlemine yarayan 'virgül' noktalama işareti ile karıştırmayın. Her iki virgülün birbiri ile karışık kullanılması geçerli bir durumdur fakat ikisini birbirinden ayırabilmek için parantezler kullanınız.

Fonksiyon bağımsız değişkenleri ve başlatıcı listelerindeki virgüllerle (Sıralama operatörü arasındaki) karmaşıklığı önlemek için parantezler kullanınız. Örnek olarak:

```
func(i, (j = 1, j + 4), k);
```

Dikkat ediniz, `func` fonksiyonu ilk bakışta gözü yanılttığı gibi 4 değil sadece 3 bağımsız değişken ile çağrılır. Bunlar `(i, 5, k)`'dır.

DEYİMLER (STATEMENTS)

Deyimler, bir program çalışırken program akışının kontrolünü belirlerler. Özel atlama (jump) ve seçme deyimlerinin yokluğunda, deyimler kaynak koddaki diziliş yerlerine göre sıralı olarak çalıştırılırlar.

Deyimler aşağıdaki gibi sınıflandırılırlar:

- Etiketli Deyimler
- İfade Deyimleri
- Seçme Deyimleri
- Yineleme Deyimleri (Döngüler)
- Atlama Deyimleri
- Birleşik Deyimler (Bloklar)

Etiketli Deyimler (Labeled Statements)

Program içerisindeki her deyim etiketlenebilir. Etiket, deyimin önüne şu şekilde yerleştirilen bir tanıtıcıdır:

```
etiket_tanitici : deyim;
```

Etiketin herhangi bir özel bildirimi yoktur. Sadece ilgili deyimi "etiketler". `etiket_tanitici` bir fonksiyon kapsamına (scope) sahiptir ve etiket aynı fonksiyonda tekrar tanımlanamaz.

Etiketler kendi isim alanlarına sahiptirler: etiket tanıtıcı, program içerisindeki herhangi başka bir tanıtıcı ile (isim olarak) çakışabilir.

Bir deyim iki nedenden dolayı etiketlenebilir:

1. Etiket tanıtıcısı, koşulsuz `goto` deyimi için bir hedef olarak işlev görür,

2. Etiket tanıtıcısı, `switch` deyimi için bir hedef olarak işlev görür. Bu amaçla, sadece `case` ve `default` etiketli deyimler kullanılır:

```
case sabit-ifade : deyim
default : deyim
```

İfade Deyimleri

Noktalı virgülün izlediği herhangi bir ifade bir ifade deyimi oluşturur:

```
ifade;
```

mikroC, *ifade*'yi değerlendirerek (hesaplayarak) bir ifade deyimini çalıştırır. Bu değerlendirmedeki tüm yanal etkiler sonraki deyimin çalıştırılmasından önce tamamlanırlar. Çoğu ifade deyimleri atama deyimleri veya fonksiyon çağrılarıdır.

null deyimi tek noktalı virgül (;) içeren özel bir durumdur. `null` deyimi hiçbir şey yapmaz. Bu nedenle mikroC sözdiziminin beklediği fakat programınızın ihtiyaç duymadığı durumlarda kullanışlıdır. Örnek olarak `null` deyimi genelde "boş" döngülerde kullanılır:

```
for (; *q++ = *p++ ;);
/* bu dongunun govdesi bir null deyimidir */
```

Seçme Deyimleri

Seçim veya akış-kontrol deyimleri belli değerleri test ederek alternatif eylem akışları arasından seçim yaparlar. C'de iki çeşit seçme deyimi vardır: `if` ve `switch`.

If Deyimi

Bir koşullu deyim uygulamak için `if` deyimi kullanılır. Söz dizimi şu şekildedir:

```
if (ifade) deyim1 [ else deyim2]
```

ifade doğru olarak değer kazanırsa, *deyim1* çalışır. Eğer *ifade* yanlış ise *deyim2* çalışır. *ifade* sonucu bir tümleşik değer olarak değerlenmelidir; aksi durumda, koşul bozuk biçimli (ill-formed) olur. *ifade*'nin çevresindeki parantezler zorunludur.

`else` anahtar sözcüğü seçimliktir. Fakat hiçbir deyim `if` ile `else` arasına gelemez.

İç-içe geçmiş if deyimleri

İç-içe geçmiş `if` deyimleri fazladan dikkat isterler. Genel kural şudur. İç-içe geçmiş koşullar en içteki koşuldan itibaren ayırılır ve her `else` solundaki (eğer varsa) ilk `if` 'e bağlanır:

```
if (ifade1) statement1
else if (ifade2)
  if (ifade3) deyim2
  else deyim3          /* bu else sozcugu if (ifade3)'e aittir */
else deyim4            /* bu else sozcugu if (ifade2)'ye aittir */
```

Not: `#if` ve `#else` ön-işlemci deyimleri (direktifleri) `if` ve `else` deyimlerine benzerdirler. Fakat çok değişik etkilere sahiptirler. Bunlar kaynak dosyası satırlarından hangisinin derlendiğini hangisinin atıldığının kontrolünü yaparlar. Daha fazla bilgi için ön-işlemci bölümüne bakınız.

Switch Deyimi

`switch` deyimi, belirli bir koşula bağlı olarak kontrolü özel bir program dalına geçirmek için kullanılır:

```
switch (ifade) {
  case sabit-ifade_1 : deyim_1;
     .
     .
     .
  case sabit-ifade_n : deyim_n;
  [ default : deyim;]
}
```

İlk olarak, *`ifade`* (koşul) değerlendirilir . Daha sonra `switch` deyimi onu mevcut *`case`* anahtar sözcüklerini izleyen tüm *`sabit-ifadeler`* ile karşılaştırır. Eğer eşleşme bulursa, switch kontrolü eşleşen duruma (case) geçirir. Bu nokta eşleşmeden sonra koşturulacak *`deyim`* noktasıdır. Unutmayın ki *`sabit-ifadeler`* değerleri hesaplandığında tamsayı olarak değer kazanmalıdırlar. Aynı değeri kazanan iki aynı *`sabit-ifade`* olamaz.

`ifade`'nin etrafındaki parantezler zorunludur.

Bir eşleme bulunması üzerine, program akışı normal devam eder. İzleyen komutlar olası bir (`case`) etiketi olsun veya olmasın doğal akışında, sırayla koşturulacaklardır. Eğer hiçbir durum (case) şarta uymazsa, `default` satırı çalıştırılır (eğer `default` etiketi belirtilmişse).

Örnek olarak, eğer `i` değeri 1 ve 3 arasında değerlere sahipse, `switch'` ten sonra sonuç daima 4 olarak dönecektir:

```
switch (i) {
  case 1: i++;
  case 2: i++;
  case 3: i++;
}
```

Başka (`case`) komutlarının çalıştırılmasından kaçınabilmek için ve değerlendirmeyi bırakıp, kontrolü `switch`'den almak için, her durumu (case) `break` ile sonlandırın.

Koşullu `switch` deyimleri iç-içe geçebilirler. Bu durumda `case` ve `default` etiketleri en içteki `switch` deyimine bağlanırlar.

`switch` ile ilgili aşağıda basit bir örnek yer almaktadır. Farz edelim ki sadece 3 farklı durumu alabilen bir değişkenimiz (0, 1, veya 2) ve her durum için bunlara karşılık gelen bir fonksiyonumuz (olay) olsun. Bu şartlarda kodu uygun yordama nasıl anahtarlama yaptırdığımızı aşağıda görelim:

```
switch (durum) {
  case 0: Lo();  break;
  case 1: Mid(); break;
  case 2: Hi();  break;
  default: Mesaj("Gecersiz durum!);
}
```

Yineleme Deyimleri

Yineleme deyimleri bir deyim kümesini döngüsel olarak koşturmamızı sağlar. C'de 3 çeşit yineleme deyimi mevcuttur: `while`, `do`, ve `for`.

While Deyimi

`while` anahtar sözcüğünü bir deyim veya deyim kümesini koşullu olarak tekrarlamak için kullanınız. `while` 'deyiminin söz dizimi:

while (*ifade*) *deyim*

İlgili `deyim` belirtilen ifade doğru olduğu sürece tekrarlı olarak çalışacaktır. Test işlemi komutlar çalıştırılmadan önce yapılır. Bu nedenle `ifade` ilk geçişte yanlış olarak sonuç verirse döngü hiç çalışmayacaktır.

ifade 'nin çevresindeki parantezler zorunludur.

Aşağıdaki örnekte `while` komutu kullanılarak iki vektörün skalar çarpımları hesaplanmıştır:

```
int s = 0, i = 0;
while (i < n) {
   s += a[ i] * b[ i];
   i++;
}
```

Dikkat ediniz ki döngünün gövdesi bir boş (null) deyim olabilir. Örnek olarak :

```
while (*q++ = *p++);
```

Do Deyimi

`do` deyimi, koşul "yanlış" olana kadar çalışacaktır. `do` deyiminin sözdizimi şu şekildedir:

```
do deyim while (ifade);
```

İlgili *ifade*'nin değeri sıfır olmadığı sürece *deyim* tekrarlı olarak çalıştırılır. Her tekrardan sonra *ifade* doğru mu, yanlış mı diye değerlendirilir. Bu nedenle ilk kontrolde ifade yanlış bulunsa bile, döngü, komutları en az bir defa çalıştırmış olacaktır.

ifade'yi çevreleyen parantezler zorunludur.

Dikkat ediniz ki; `do`, C dilinde noktalı virgül (;) ile sonlanan tek kontrol yapısıdır. Diğer kontrol yapıları ise bir deyim ile sonlanırlar yani bu demektir ki noktalı virgül ile biten bir deyim veya kapama kaşlı parantezi içeren deyim grubu ile sonlanırlar.

Aşağıda `do` komutu kullanılarak iki vektörün skalar çarpımları hesaplanmıştır:

```
s = 0; i = 0;
do {
   s += a[ i] * b[ i];
   i++;
} while (i < n);
```

For Deyimi

`for` deyimi tekrarlı bir döngüdür. Söz dizimi şu şekildedir:

```
for ([ baslama-ifadesi]; [ şart-ifadesi]; [ artım-ifadesi]) deyim
```

Döngünün ilk tekrarından önce, *başlama-ifadesi* başlama değerlerini döngü için kurar. *başlama-ifadesi* içerisindeki tanımlamaları gözardı edemezsiniz.

şart-ifadesi bloğa ilk girişten önce kontrol edilir. *deyim, şart ifadesi* yanlış olana kadar tekrarlı çalıştırılır. Döngünün her tekrarından sonra, *artım-ifadesi* bir döngü sayacını arttırır (veya azaltır). Bu nedenle, i++ fonksiyonel olarak ++i ile aynıdır.

Tüm ifadeler seçimliktir. Eğer şartlı-ifade boş bırakılırsa daima doğru olarak varsayılır. Böylelikle, "boş" bir `for` deyimi, genelde bir sonsuz döngü oluşturmak için kullanılır:

```
for ( ; ; ) {...}
```

Bu döngüden tek çıkış yolu `break` deyimini kullanmaktır.

Aşağıda iki vektörün skaler çarpımını `for` deyimi kullanılarak gösterilmiştir:

```
for (s = 0, i = 0; i < n; i++) s += a[i] * b[i];
```

Aynı zamanda şu şekilde yapabilirsiniz:

```
/* gecerli ama pek sevimsiz bir yol */
for (s = 0, i = 0; i < n; s += a[i] * b[i], i++);
```

fakat bu bir kötü programlama stili olarak görülür. Geçerli olmasına rağmen, toplamın hesaplanması arttırılan ifadenin bir parçası olmamalıdır, çünkü, bir döngünün hizmetinde değildir. Dikkat ediniz ki döngü gövdesi için boş (null) deyim kullandık.

Atlama Deyimleri (Jump Statements)

Bir atlama deyimi çalıştırıldığında, kontrolü koşulsuz bir şekilde transfer eder. mikroC'de bu şekilde dört deyim mevcuttur: `break`, `continue`, `goto`, ve `return`.

Break Deyimi

Bazen, döngüyü kendi gövdesi içerisinde kesmek isteyebilirsiniz. `break` deyimini kullanarak kontrol'ü en içteki `switch`, `for`, `while` veya `do` bloğunu takip eden ilk deyime geçirebilirsiniz.

`Break` genelde `switch` deyimleri içersinde ilk pozitif eşleşmeden sonra çalışmayı durdurmak için kullanılır. Örnek olarak:

```
switch (durum) {
  case 0: Lo();  break;
  case 1: Mid(); break;
  case 2: Hi();  break;
  default: Mesaj("geçersiz durum!");
}
```

Continue Deyimi

`continue` deyimini döngüler (`while, do, for`) içerisinde "o turun geri kalanını atlamak " için kullanabilirsiniz. Kontrol, döngü kurgusunu sağlayan en içteki parantezin sonuna geçer. Bu noktada, döngü devam koşulu tekrar değerlendirilir. Yani eğer döngü devam koşulu *doğru* ise, `continue` bir sonraki tekrarı talep edecektir.

Goto Deyimi

`goto` deyimi bir yerel etikete şartsız atlama için kullanılır. Etiketler hakkında daha fazla bilgi için "Etiket Deyimleri" bölümüne bakınız. `goto` nun söz dizimi:

```
goto etiket_tanıtıcısı;
```

Bu sözdizimi kontrolü, `etiket_tanıtıcısı`'nın belirttiği yere taşır. `etiket_tanıtıcısı` goto deyimi ile aynı fonksiyon içerisindeki bir etiketin ismi olmalıdır. goto satırı etiketten önce veya sonra yer alabilir.

`goto`'yu iç-içe geçmiş kontrol yapılarının herhangi düzeyinden dışarı çıkmak için kullanabilirsiniz. Fakat `goto` blokun başlatmaları atlanarak blok içerisine atlamak için kullanılamaz. Mesela, döngü gövdesi içerisine atlama, vs. uygun değildir.

Okunaklı yapılandırılmış programlarda, algoritma `goto` deyimi olmadan da gerçekleştirilebildiğinden, `goto` deyiminin kullanılması pek tavsiye edilmez. `goto` deyiminin en uygun uygulamalarından biri de derince iç-içe geçmiş kontrol yapılarından çıkmaktır:

```
for (...) {
  for (...) {
  ...
    if (cok_kotu_durum) goto hata;
  ...
  }
}
 .
 .
 .
hata: /* hata isleme kodu */
```

Return Deyimi

`return` deyimi, mevcut fonksiyondan asıl çağıran yordama, opsiyonel olarak bir değer döndürerek geri dönmek için kullanılır. Söz dizimi şu şekildedir:

```
return [ ifade] ;
```

Bu *`ifade`*'yi değerlendirecek ve sonucu döndürecektir. Gerekli olursa, dönen değer otomatik olarak beklenen fonksiyon dönüş tipine dönüştürülecektir. *`ifade`* seçimliktir; eğer ihmal edilirse fonksiyon bellekten rastgele bir değeri döndürecektir.

Not: `void` tipi fonksiyonlar içerisindeki `return` deyimi bir *`ifade`*'ye sahip olamaz. Aslında, eğer fonksiyon gövdesinin son deyimi ise, `return`'u tümden kaldırabilirsiniz.

Birleşik Deyimler (Bloklar)

Bir birleşik deyim veya blok, çift kaşlı parantez ile sınırlandırılmış deyimler grubudur. Söz-dizimsel olarak, bir blok tek bir deyim olarak göz önüne alınabilir. Fakat aynı zamanda tanıtıcıların kapsamlarının belirlenmesinde önemli rol oynar. Bir blok içerisinde bildirilen bir tanıtıcının kapsamı bildirim noktasında başlar ve kapatma parantezi ile sonlanır. Bloklar bellek limiti dahilinde herhangi bir derinliğe kadar iç-içe kullanılabilirler.

Örnek olarak, `for` döngüsü kendi gövdesinde sadece bir deyim beklemektedir, bu sorunu aşabilmek için birleşik deyimler kullanabiliriz:

```
for (i = 0; i < n; i++) {
  int temp = a[ i] ;
  a[ i]  = b[ i] ;
  b[ i]  = temp;
}
```

Dikkat ediniz ki, diğer deyimlerin aksine, birleşik deyimler noktalı virgül (;) ile sonlandırılamaz, yani kapama parantezinden sonra hiçbir zaman noktalı virgül yer almaz.

ÖN-İŞLEMCİ (PREPROCESSOR)

Ön-işlemci, kaynak kodunu derleme için hazırlayan bir tümleşik metin işlemcisidir. Ön-işlemci şunları sağlar:

- belirtilen bir dosyadan kod içindeki belirli bir noktaya metin eklenmesi,
- özel sözlüksel sembollerin diğer sembollerle değiştirilmesi,
- kodun belli kısımlarını koşullu olarak çıkartan veya ekleyen şartlı derleme.

Dikkat ediniz ki, ön-işlemci, metni tek tek karakter düzeyinde değil dizgecik (token) düzeyinde analiz etmektedir. Ön-işlemciler, ön-işlemci direktifleri ve ön-işlemci operatörleri yoluyla denetlenirler.

Ön-işlemci Direktifleri

Kaynak kod içerisinde # işareti ile başlayan herhangi bir satır; bir karakter dizisi literali içerisinde, bir karakter sabiti içerisinde veya bir yorum içerisinde gömülü olmadığı müddetçe, bir *ön-işlemci direktifi* (veya kontrol satırı) olarak alınır. Baştaki # işaretinin başında veya sonunda beyaz boşluklar olabilir (yeni satıra geçmemek şartıyla).

`null` direktifi, sadece bir tek # karakterini içeren bir satırdan oluşur. Bu satır daima görmezlikten gelinilir.

Ön-işlemci direktifleri genellikle kaynak kodunun en başına yerleştirilir. Fakat geçerli olarak program içerisinde herhangi bir noktada da görünebilir. mikroC ön-işlemcisi, ön-işlemci direktiflerini algılar ve içlerinde gömülü olan dizgecikleri (token) ayırır. Direktif, kendi bildiriminden program dosyasının sonuna kadar etkilidir.

mikroC standart ön işlemci direktiflerini destekler:

```
# (null direktifi)        #if
#define                   #ifdef
#elif                     #ifndef
#else                     #include
#endif                    #line
#error                    #undef
```

Not: `#pragma` direktifi yapım aşamasındadır.

Ters Bölü İşareti ile Satır Devam Ettirme

Eğer direktifi çoklu satırlara bölme ihtiyacınız varsa, bunu satır sonuna bırakacağınız bir ters bölü (\) işareti ile gerçekleştirebilirsiniz:

```
#define MACRO  Bu satir asagidaki \
                satirdan devam etmektedir.
```

Makrolar

Makro'lar; derlemeden önceki aşamada dizgeciklerin (token) yenisi ile değiştirilmesi için, biçimsel veya biçimsel olmayan fonksiyon parametreleri benzeri parametrelerle veya parametresiz, bir mekanizma sağlar.

Makroları ve Makro Açılımlarını Tanımlama

`#define` direktifi bir makro tanımlar:

```
#define makro_taniticisi <dizgecik_dizisi>
```

Kaynak kod içerisinde bu kontrol satırını takip eden her *`makro_tanıtıcısı`* varlığı, aynı yerde, bazen boş da olabilen *`dizgecik_dizisi`* ile yer değiştirecektir. (bazı istisnalar mevcuttur ve bunlar daha sonra belirtilecektir). Bu yer değiştirmeler *makro açılımları* olarak bilinirler. *`dizgecik_dizisi`* bazen makronun gövdesi olarak çağırılır. Boş dizgecik dizisi, kaynak koddan ilgili her makro tanıtıcının silinmesi ile sonuçlanır.

Ön işlemci direktifini sonlandırmak için noktalı virgüle (;) ihtiyaç duyulmaz. Dizgecik dizisi içerisinde bulunan herhangi bir karakter, noktalı virgül dahil, makro açılımı içerisinde görünecektir. *`dizgecik_dizisi`*, önünde ters-bölü (\) bulundurmayan ilk yeni satır karakteri ile sonlanır. Herhangi bir beyaz-boşluk dizisi, dizgecik dizisi içerisindeki yorumlar dahil, bir tek boşluk karakteri olarak yerleştirilir.

Her makro açılımından sonra yeni açılımı yapılan metin üzerinde tekrar bir tarama yapılır. Bu, iç-içe geçmiş makroların kullanılmasına izin verir: Açılımı yapılmış metin, değiştirmeye uygun makro tanıtıcıları içerebilir. Ancak, eğer makro açılımı bir ön-işlemci direktifine benzer şekilde açılır ise bu direktif ön-işlemci tarafından tanınmayacaktır. Makro tanıtıcısı bir literal karakter dizisi, karakter sabitleri veya kod yorumları içerisinde bulunuyorsa açılımı yapılmaz.

Bir makronun kendi açılımı sırasında tekrar açılımı yapılmaz. (bu nedenle `#define MACRO MACRO` sonsuza dek tekrar tekrar açılmayacaktır).

Şu örneğe bakalım:

```
/* Asagida bazi basit makrolar bulunmaktadir : */
#define ERR_MSG "Erim disinda!"
#define EVERLOOP for( ; ; )

/* Bunu su sekilde de kullanabiliriz: */

main() {
  EVERLOOP {
    ...
    if (error) {Lcd_Out_Cp(ERR_MSG); break;}
    ...
  }
}
```

Daha önceden tanımlanmış bir makro tanıtıcısını tekrar tanımlamaya kalkışmak, yeni tanımlamanın dizgecikleri eski tanımlamanın dizgecikleri ile (token) bire bir uymadığı müddetçe hata üretecektir. Diğer başlık dosyalarında başka tanımlamaların olabileceği tahmin ediliyorsa, tercih edilen strateji şudur:

```
#ifndef BLOK_BOYUTU
  #define BLOK_BOYUTU 512
#endif
```

Eğer `BLOK_BOYUTU` ilgili durumda tanımlı ise orta satır ihmal edilebilir. Eğer `BLOK_BOYUTU` ilgili durumda tanımlı değilse, orta satır onu tanımlamak için değerlendirilir.

Parametreli Makrolar

Aşağıdaki söz dizimi parametreleri olan bir makronun tanımlanması için kullanılır:

```
#define makro_taniticisi(<arguman_listesi>) dizgecik_dizisi
```

Dikkat ediniz ki, `makro_taniticisi` ile "(" arasında beyaz boşluk olamaz. Seçimlik olan *`arguman_listesi`* virgüller ile ayrılan tanıtıcıların listesidir. Bir C fonksiyonunun bağımsız değişken listesinden pek de farklı değildir. Her virgül ile sınırlanmış tanıtıcı, biçimsel bağımsız değişken veya yer-tutucu (placeholder) işlevi görür.

Makrolar, takip eden kaynak kodun içinde şu şekilde çağırılırlar:

```
makro_taniticisi(<gercek_arguman_listesi>)
```

Söz dizimi bir fonksiyon çağrısı ile aynıdır. Gerçekte, birçok standart C kütüphane "fonksiyonları" makrolar olarak gerçeklenmişlerdir. Ancak bazı önemli anlamsal farklılıklar mevcuttur.

Seçimlik olan *`gercek_arguman_listesi`*, `#define` satırının *`arg_list`*'inde bulunan biçimsel argumanlarla aynı sayıda, virgülle sınırlandırılmış dizgecik dizilerini içermelidir-yani her biçimsel argumana karşılık bir gerçek arguman bulunmalıdır. Eğer iki liste içerisindeki arguman sayıları farklı olursa bir hata durumu bildirilir.

Bir makro çağrısı, iki yerine koyma işlemi ile sonuçlanır. İlk önce, makro tanıtıcısı ve parantezler ile kapatılmış bağımsız değişkenler (argumanlar) yerine, dizgecik (token) dizisi yerleştirilir. Sonra, dizgecik dizisi içerisinde ortaya çıkan her biçimsel (yani tanımsal) bağımsız değişken yerine, gerçek*`_arguman_listesi`*'nde görünen gerçek bağımsız değişken koyulur. Basit makro tanımlarında olduğu gibi, açılımı yapılmaya uygun herhangi bir gömülü makro tanıtıcısını saptayabilmek için yeniden bir tarama daha yapılır.

Örnek:

```
/* kendi iki argumanindan (yani bagimsiz degiskeninden) buyuk
olanini donduren basit bir makro : */

#define _MAX(A, B) ((A) > (B)) ? (A) : (B)

// Haydi makromuzu cagiralim:
x = _MAX(a + b, c + d);

/* On-islemci yukaridaki satiri
x = ((a + b) > (c + d)) ? (a + b) : (c + d)
sekline donusturecektir */
```

Makro gövdesi içerisinde her bağımsız değişkenin etrafına parantezler koymak tavsiye edilir - Bu oluşabilecek operatör önceliği sorunlarından kaçınmamızı sağlar.

Makroları Tanımsızlaştırma

Bir makroyu `#undef` direktifini kullarak tanımsız yapabilirsiniz.

```
#undef makro_taniticisi
```

`#undef` direktifi önceki dizgecik dizilerini de `makro_taniticisi`'ndan ayırır; Makro tanımı unutturulur. Ve `makro_taniticisi` tanımsız hale getirilir. `#undef` satırlarının içerisinde hiçbir makro açılımı yapılmaz.

Asıl tanımlamaya bakmaya gerek kalmadan, tanımlı veya tanımsız olma durumu bir tanıtıcı için önemli bir özelliktir. `#ifdef` ve `#ifndef` koşul direktifleri, herhangi bir tanıtıcının şu anda tanımlı olup olmadığını test etmek için kullanılır ve derleme işleminin birçok özelliğinin kontrolü açısından esnek bir mekanizma sağlar.

Bir makro tanıtıcısını tanımsızlaştırdıktan sonra, `#define` kullanarak (istersek aynı veya istersek de farklı dizgecik dizisi ile) tekrar tanımlı hale getirebiliriz.

Dosya Eklenmesi

Ön-işlemci direktifi olan `#include` başlık dosyalarını (`.h` uzantılıdır) kaynak kod içine koyar. Ön-işlemcinin, kaynak dosyalarını (.c uzantılıdır) dahil edeceğine güvenmeyiniz. Daha fazla bilgi için "Projeler" bölümüne bakınız.

`#include` direktifinin iki farklı formatta söz dizimi vardır:
`#include <başlık_ismi>`
`#include "başlık_ismi"`

Ön-işlemci `#include` satırını siler ve kaynak kod içindeki bu noktaya başlık dosyasını, tüm metinleriyle birlikte yerleştirir. `#include'` un yerleşimi bundan dolayı eklenen dosya içinde herhangi bir tanıtıcının kapsam (scope) ve süresini (duration) etkileyebilir.

İki format arasındaki fark, eklenecek dosyanın bulunması için yapılacak arama algoritmasında yatar.

Eğer `#include` direktifi `<başlık_ismi>` versiyonu ile kullanılmış ise, arama aşağıdaki yerlerde ve şu düzen içinde gerçekleştirilir:

1. mikroC kurulum dizini > "include" dizini,
2. kendi özel arama yolunuz.

`"başlık_ismi"` versiyonu kullanıcı tarafından sağlanan bir ekleme dosyası belirtir. MikroC başlık dosyası için aşağıdaki yerlerde ve şu düzen içinde bakacaktır:

1. proje dizini (proje dosyasını içeren dizin `.ppc`),
2. mikroC kurulum dizini > "include" dizini,
3. kendi özel arama yolunuz.

Açıkça Arama Yolu Belirtmek

Eğer `başlık_ismi` içine açıkça bir yol yerleştirmişseniz, sadece bu yol aranacaktır. Örnek olarak:

```
#include "C:\my_files\test.h"
```

Not: `#include` direktifinin üçüncü bir versiyonu daha vardır. Nadiren kullanılır. Bu versiyon şunu varsayar: `#include`'dan sonra ne < işareti ne de " işareti `#include` sözcüğünü izleyen ilk beyaz boşluk olmayan karakter olarak görünmez:

```
#include makro_taniticisi
```

Makro tanıtıcısının açılımının, <başlık_ismi> yada "başlık_ismi" geçerli başlık dosyası biçimlerinden birine yapılacağını varsayar.

Ön-işlemci Operatörleri

Diyez işareti (#), bir satır üzerinde ilk beyaz-boşluk olmayan karakter olarak ortaya çıktığında ön-işlemci direktifi olur. Öte taraftan, `#` ve `##` simgeleri ön-işlemci tarama aşamasında operatör yerleştirmesi ve birleştirmesi işlemini yaparlar.

Operatörü

C ön-işlemcide, çift tırnaklar arasındaki karakterler bir dizgecik olarak görülürler ve içerikleri analiz edilmez. Bu şu anlama gelmektedir: tırnak içindeki makro isimlerin açılımı yapılmaz.

Bir bağımsız değişkenin adına, ön-işlemci sonucunda ihtiyaç duyuyorsanız (tırnak içerisindeki bir tam karakter dizisi olarak), makro gövdesi içerisinde # operatörünü kullanabilirsiniz. Tanımlama içerisinde biçimsel bir makro bağımsız değişkeninin önüne, gerçek bağımsız değişkenin adını yerleştirme işleminden sonra bir diziye dönüştürme maksadıyla koyulmuş olabilir.

Örnek olarak, değişken isim ve değerini LCD üzerinde göstermek için bir makro `LCD_PRINT` oluşturalım:

```
#define LCD_PRINT(val)  Lcd_Out_Cp(#val ": "); \
                        Lcd_Out_Cp(IntToStr(val));
```

(unutmayın ki ters bölü işareti satıra-devam etmenin sembolüdür!)

Aşağıdaki kod:,

```
LCD_PRINT(temp)
```

ön-işlem ile şu şekile dönüşür:

```
Lcd_Out_Cp("temp" ": "); Lcd_Out_Cp(IntToStr(temp));
```

Operatörü

`##` operatörü dizgeciklerin birbirlerine tutturulması için kullanılır: iki dizgeciği aralarına ## koyarak tutturabilirsiniz (veya birleştirebilirsiniz). Ön-işlemci beyaz boşluğu ve ## simgesini siler, ayrı dizgecikleri yeni bir dizgecik halinde birleştirir. Bu genelde tanıtıcıları oluşturmak için kullanılır.

Örnek olarak, iki dizgeciği bir tanıtıcı içerisinde yapıştırmak için `SPLICE` makro' sunu tanımlayabiliriz.

```
#define SPLICE(x,y) x ## _ ## y
```

Şimdi, `SPLICE(cnt, 2)` çağrısı `cnt_2` tanıtıcı haline açılır.

Not: mikroC eski uygun olmayan şu tip dizgecik tutturma metodlarını desteklemez: `(l/**/r)`.

Koşullu Derleme

Koşullu derleme direktifleri, farklı koşturma ortamları içerisinde kaynak programları daha kolay değiştirilebilir, daha kolay derlenebilir yapmak için kullanılır. mikroC, uygun kaynak kod satırlarının boş bir satır ile değiştirilmesi yoluyla koşullu derlemeleri destekler.

Tüm koşullu derleme direktiflerinin kaynak kod içerisinde veya başlamış oldukları ekleme dosyası içerisinde derlenmiş olmaları gerekir.

#if, #elif, #else, ve #endif Direktifleri

#if, #elif, #else, ve #endif şartlı direktifleri C'nin alışıldık koşullu deyimleri ile çok benzer bir şekilde çalışırlar. Eğer #if den sonra yazdığınız değer sıfır olmayan bir değer ise #if'i takip eden satır grubu işlem ünitesi içerisine alıkonulmuş olur.

Sözdizimi:

```
#if sabit_ifade_1
<bolum_1>

[ #elif sabit_ifade_2
<bolum_2>]
   ...
[ #elif sabit_ifade_n
<bolum_n>]

[ #else
<son_bolum>]

#endif
```

Kaynak dosyası içerisindeki her #if direktifi bir kapama #endif direktifi ile muhakkak eşleşmelidir. İstenilen sayıda #elif direktifleri #if ile #endif arasına konabilir. Fakat en fazla bir tane #else direktifine izin verilir. #else direktifi, eğer var ise, #endif direktifinden önceki son direktiftir.

Bölümler, derleyici veya ön-işlemci için anlamlı herhangi bir program metni olabilir. Ön-işlemci #if veya #elif direktiflerinden sonra bir doğru (sıfır olmayan) sabit ifade buluncaya kadar *sabit_ifade*'yi değerlendirerek bir bölüm seçecektir. *sabit_ifade*'ler makro açılımına tabidirler.

Eğer sabit-ifadenin tüm oluşumları yalnış (false) ise veya hiçbir #elif direktifi yoksa, ön-işlemci #else tümcesinden sonraki metin bloğunu seçecektir. Eğer #else tümcesi yoksa ve #if bloğu içerisindeki *sabit_ifade*'nin tüm örnekleri yalnış ise, hiçbir bölüm başka işlem için seçilmeyecektir.

Herhangi bir işlenmiş *bölüm* daha fazla koşullu ifadeler (herhangi bir derinlikte iç-içe geçmiş) içerebilir. Her iç-içe geçmiş `#else`, `#elif`, veya `#endif` direktifi en yakın `#if` direktifine aittir.

Önceki incelemenin net sonucu şudur: sadece bir kod *bölüm*'ü (ki boş da olabilir) derlenecektir.

#ifdef ve #ifndef direktifleri

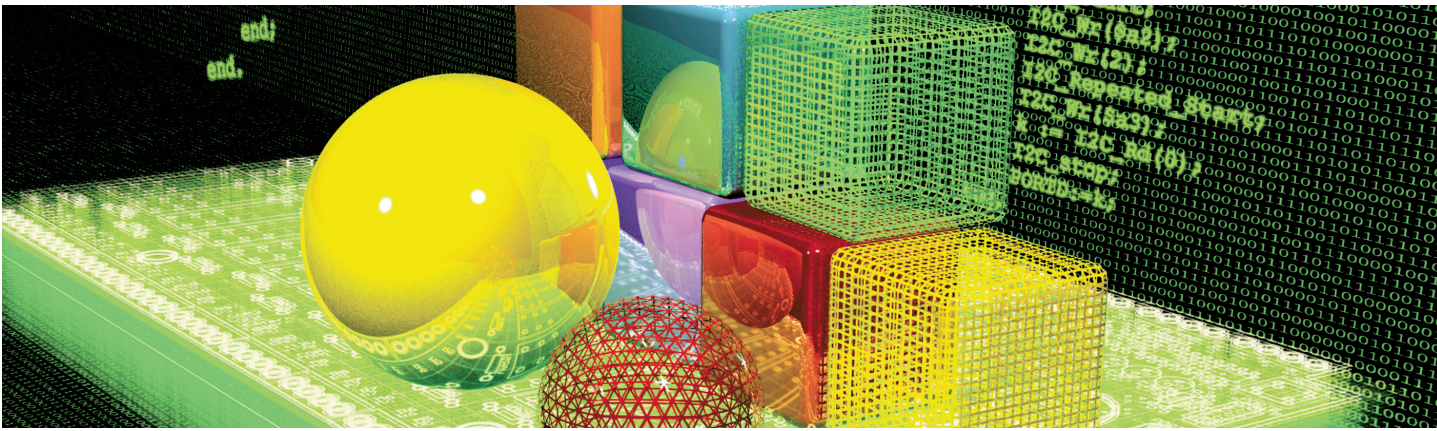
`#ifdef` ve `#ifndef` direktiflerini `#if`'in kullanılabildiği herhangi bir yerde kullanabilirsiniz. `#ifdef` ve `#ifndef` şartlı direktifleri bir tanıtıcının şu anda tanımlı olup olmadığını test etmenizi sağlar.

```
#ifdef tanitici
```

Eğer *tanitici* tanımlı ise yukaridaki satir `#if 1` ile aynı etkiyi gösterir. Eğer tanımlanmamışsa `#if 0` ile aynı etkiyi gösterir. Diğer direktif olan `#ifndef`, zıt sonuçlar üreterek "tanımlanmamış" koşullar için doğruluğu test eder.

Dikkat ediniz ki NULL (yani hiçlik) olarak tanımlanmış bir tanıtıcı tanımlanmış olarak göz önüne alınır.

BÖLÜM 5



mikroC Kütüphaneleri

mikroC uygulamalarınızı hızlı ve kolay geliştirmeniz için birtakım hazır yordamlar ve kütüphaneler bulundurmaktadır. ADC, CAN, USART, SPI, I2C, one-Wire, LCD, PWM, RS-485, Seri Erhernet, Toshiba GLCD, Port Genişletici, Seri GLCD, sayısal biçimlendime, bit işleme uygulamaları için geliştirilmiş hazır kütüphanelerin yanında birçok uygulama ve kullanmaya hazır kod örneklerini de mikroC derleyicisinin içerisinde bulabilirsiniz. Bu alt yapı PIC mikrodenetleyicisi kullanıcıları için eşsiz bir kaynaktır.

YERLEŞİK YORDAMLAR (BUILT-IN ROUTINES)

mikroC derleyicisi, bir dizi kullanışlı yerleşik fonksiyonlar içerir. Yerleşik yordamlar herhangi bir başlık dosyasına ihtiyaç duymazlar. Projenin herhangi bir yerinde kullanılabilirler.

Yerleşik yordamlar "kodiçi" (inline) gerçekleştirilmişlerdir yani kodları tam çağırıldıkları yerde yerleştirilir. Bu nedenle bu yordamların çağrıları, iç-içe dallanma limitinde hesaba katılmaz. Tek istisna gerçek C yordamları olan Vdelay_ms ve Delay_Cyc yordamlarıdır.

Not: Lo, Hi, Higher ve Highest fonksiyonları artık derleyici içerisinde gömülü değildirler. Eğer bu fonksiyonlar kullanılmak isteniyorsa, projeye built_in.h dahil edilmelidir.

Lo
Hi
Higher
Highest

Delay_us
Delay_ms
Vdelay_ms
Delay_Cyc
Clock_Khz
Clock_Mhz

Lo

Yapısı	`unsigned short Lo(long number);`
Dönüş	`number`'ın alt 8-bitini (0..7 bitleri) döndürür.
Tanımı	Fonksiyon `number`'in alt baytını döndürür. Fonksiyon `number`'ın bitlerini yorumlamaz, sadece kayıtçıda bulduğu 8 biti döndürür. Bu "kodiçi" (inline) bir yordamdır. Kod çağırıldığı yerde üretilir. Bu nedenle çağrı, iç-içe dallanma limitine karşı hesaba katılmaz
Gereklilikler	Bağımsız değişkenler sayısal tipte olmalıdırlar (Örneğin: Aritmetik tipler ve işaretçiler).
Örnek	`d = 0x1AC30F4;` `tmp = Lo(d);  // = 0xF4`

Hi

Yapısı	`unsigned short Hi(long number);`
Dönüş	`number`'ın alt baytdan bir sonraki baytını döndürür (8..15 bitleri).
Tanımı	Fonksiyon `number` 'ın alt byte'dan bir sonraki baytını döndürür. Fonksiyon number'ın bitlerini yorumlamaz, sadece kayıtçıda bulduğu 8 biti döndürür. Bu "kodiçi" bir yordamdır. Kod çağırıldığı yerde üretilir. Bu nedenle çağrı, iç-içe dallanma limitine karşı hesaba katılmaz
Gereklilikler	Bağımsız değişkenler sayısal tipte olmalıdırlar (Örneğin: Aritmetik tipler ve işaretçiler).
Örnek	`d = 0x1AC30F4;` `tmp = Hi(d);  // = 0x30`

Higher

Yapısı	`unsigned short Higher(long number);`
Dönüş	`number`'ın üst baytdan bir önceki baytını (16..23 bitleri) döndürür.
Tanımı	Fonksiyon `number` 'ın üst byte'dan bir önceki baytını döndürür. Fonksiyon `number`'ın bitlerini yorumlamaz, sadece kayıtçıda bulduğu 8 biti döndürür. Bu "kodiçi" bir yordamdır. Kod çağırıldığı yerde üretilir. Bu nedenle çağrı, iç-içe dallanma limitine karşı hesaba katılmaz
Gereklilikler	Bağımsız değişkenler sayısal tipte olmalıdırlar (Örneğin: Aritmetik tipler ve işaretçiler).
Örnek	`d = 0x1AC30F4;` `tmp = Higher(d);  // = 0xAC`

Highest

Yapısı	`unsigned short Highest(long number);`
Dönüş	`number`'ın en üst byte'ını (24..31 bitleri) döndürür.
Tanımı	Fonksiyon `number`'ın en üst baytını döndürür. Fonksiyon `number`'ın bitlerini yorumlamaz, sadece kayıtçıda bulduğu 8 biti döndürür Bu "kodiçi" bir yordamdır. Kod çağırıldığı yerde üretilir. Bu nedenle çağrı, iç-içe dallanma limitine karşı hesaba katılmaz
Gereklilikler	Bağımsız değişkenler sayısal tipte olmalıdırlar (Örneğin: Aritmetik tipler ve işaretçiler).
Örnek	`d = 0x1AC30F4;` `tmp = Highest(d);  // = 0x01`

Delay_us

Yapısı	`void Delay_us(const time_in_us);`
Tanımı	Mikrosaniyeler düzeyinde (`time_in_us`) bir yazılımsal gecikme oluşturur. Fonksiyonun parametresi bir sabittir ve uygulanabilir sabit süre aralığı osilatör frekansına bağlıdır.
Örnek	`Delay_us(10);  /* 10 mikro-saniye durma */`

Delay_ms

Yapısı	`void Delay_ms(const time_in_ms);`
Tanımı	Milisaniyeler düzeyinde (`time_in_ms`) bir yazılımsal gecikme oluşturur. Fonksiyonun parametresi bir sabittir ve uygulanabilir sabit süre aralığı osilatör frekansına bağlıdır.
Örnek	`Delay_ms(1000);  /* Bir saniye durma */`

Vdelay_ms

Yapısı	`void Vdelay_ms(unsigned time_in_ms);`
Tanımı	Milisaniyeler düzeyinde (`time_in_ms`) bir yazılımsal gecikme oluşturur. Fonksiyonun parametresi bir değişkendir ve oluşturulan gecikme `Delay_ms` tarafından gerçekleştirilen gecikme kadar hassas değildir.

Örnek	`pause = 1000;`
	`// ...`
	`Vdelay_ms(pause);  // Yaklasik bir saniye durma`

Delay_Cyc

Yapısı	`void Delay_Cyc(char Cycles_div_by_10);`
Tanımı	MCU saatine bağlı bir gecikme oluşturur. Gecikme , MCU döngüsü cinsinden hesaplanır ve fonksiyon parametresinin 10 katı süre kadardır. Giriş parametresi 3 ile 255 arasında olmalıdır. Dikkat ediniz ki `Delay_Cyc` yerleşik bir yordamdan ziyade bir kütüphane fonksiyonudur. Uyumluluk açısından bu başlık altında verilmiştir.
Örnek	`Delay_Cyc(10);  /* 100 MCU dongusu durma (pause) */`

Clock_Khz

Yapısı	`unsigned Clock_Khz(void);`
Dönüş	KHz biriminde MCU saat'i. En yakın tam sayıya yuvarlanmıştır.
Tanımı	MCU saat'i KHz biriminde döndürür. Sonuç en yakın tam sayıya yuvarlanmıştır.
Örnek	`clk = Clock_Khz();`

Clock_Mhz

Yapısı	`unsigned Clock_Mhz(void);`
Dönüş	MHz biriminde MCU saat'i. En yakın tam sayıya yuvarlanmıştır.
Tanımı	MCU saat'i MHz biriminde döndürür. Sonuç en yakın tam sayıya yuvarlanmıştır.
Örnek	`clk = Clock_Mhz();`

KÜTÜPHANE YORDAMLARI (LIBRARY ROUTINES)

mikroC, PIC mikrodenetleyicilerinin ve birimlerinin kullanımını basitleştirmek için bir kütüphane seti sağlamaktadır. Kütüphane fonksiyonları herhangi bir başlık dosyasına ihtiyaç duymaz; siz onları projenizde istediğiniz yerde kullanabilirsiniz.

Mevcut kütüphaneler şunlardır:

Donanıma/PIC'e - özel Kütüphaneler

- ADC Kütüphanesi
- CAN Kütüphanesi
- CANSPI Kütüphanesi
- Compact Flash Kütüphanesi
- EEPROM Kütüphanesi
- Ethernet Kütüphanesi
- SPI Ethernet Kütüphanesi
- Flash Bellek Kütüphanesi
- Grafik LCD Kütüphanesi
- T6963C Grafik LCD Kütüphanesi
- I²C Kütüphanesi
- Tuştakımı Kütüphanesi
- LCD Kütüphanesi
- Özel LCD Kütüphanesi
- LCD8 Kütüphanesi
- Manchester Kodu Kütüphanesi
- Multi Media Kart Kütüphanesi
- Tek-tel Kütüphanesi
- PS/2 Kütüphanesi
- PWM Kütüphanesi
- RS-485 Kütüphanesi
- Yazılımsal I²C Kütüphanesi
- Yazılımsal SPI Kütüphanesi
- Yazılımsal UART Kütüphanesi
- Ses Kütüphanesi
- SPI Kütüphanesi
- USART Kütüphanesi
- USB HID Kütüphanesi
- Util Kütüphanesi
- SPI Grafik LCD Kütüphanesi
- Port Genişletici Kütüphanesi

- SPI LCD Kütüphanesi
- SPI LCD8 Kütüphanesi
- SPI T6963C Grafik LCD Kütüphanesi

Standart ANSI C Kütüphaneleri

- ANSI C Ctype Kütüphanesi
- ANSI C Math Kütüphanesi
- ANSI C Stdlib Kütüphanesi
- ANSI C String Kütüphanesi

Diğer Kütüphaneler

- Dönüşüm Kütüphanesi
- Trigonometri Kütüphanesi
- sprint Kütüphanesi
- Setjmp Kütüphanesi
- Zaman Kütüphanesi

ADC Kütüphanesi

ADC (Analog to Digital Converter - Analog'dan Dijital'e Çevirici) birimi sadece yonga üzerinde ADC olan PIC MCU'lar ile birlikte kullanılabilir. Kütüphane fonksiyonu `ADC_Read` sizin ADC birimi ile daha rahat çalışmanızı sağlar.

Adc_Read

Yapısı	`unsigned Adc_Read(char channel);`
Dönüş	Belirtilmiş ADC kanalından 10 bit işaretsiz değeri oku.
Tanımı	PIC'in iç ADC modülünü, RC saat ile çalışması için başlangıç durumuna getirir. Saat AD (analog-dijital) çevrim için gerekli olan süreyi ayarlar (en az 12TAD). `channel` parametresi hangi ADC kanalından bilginin yakalanacağını gösterir. Kanal-pin eşlemesi için uygun PIC MCU dokümantasyonuna bakınız.
Gereklilikler	PIC MCU yonga üzerinde yerleşik ADC birimi olmalıdır. (PIC16/18 serilerinin çoğunda yerleşik ADC vardır). ADC biriminin kullanımı hakkında teknik bilgiler için o MCU'ya (yani mikrodenetleyiciye) ait teknik dokümana başvurmalısınız. Fonksiyonu kullanmadan önce, TRISA'nın uygun bitlerini giriş olarak yapılandırılmış olduğunuzdan emin olunuz. Aynı zamanda, istenen pinleri analog giriş olarak yapılandırınız ve referans voltajını (Vref) ayarlayınız.

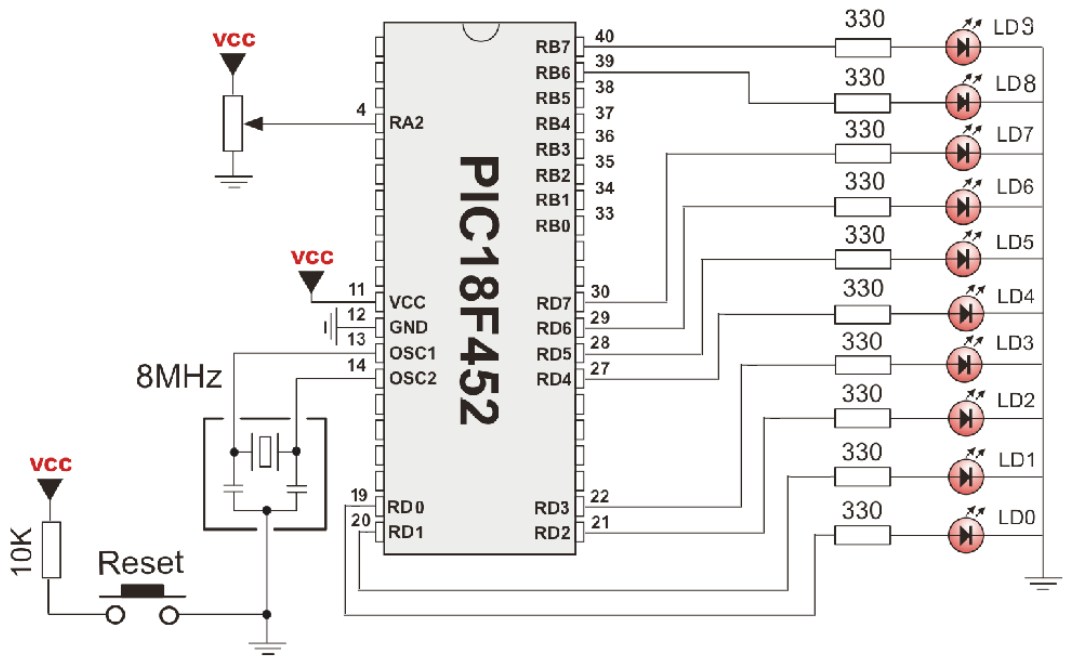
Örnek	`unsigned tmp;`
	`...`
	`tmp = Adc_Read(1);  /* kanal 1'den analog degeri oku */`

Library Örnek

Bu kod parçası analog değeri 2. kanal'dan (RA2) alır; karşılığı olan sayısal değerin ilk 8 bit'ini PORTD'ye, sonraki 2 bit'ini ise PORTB'ye gönderir.

```
unsigned temp_res;
void main() {
  ADCON1 = 0x80;    // Analog girisleri ve Vref'i yapilandir
  TRISA  = 0xFF;    // PORTA giris
  TRISB  = 0x3F;    // RB7 ve RB6 pinleri cikis
  TRISD  = 0;       // PORTD cikis
  do {
    temp_res = Adc_Read(2);      // AD donusum sonucunu al
    PORTD = temp_res;            // Alt 8-biti PORTD'ye yolla
    PORTB = temp_res >> 2;       // En ust 2 biti RB6 ve RB7'ye yolla
  } while(1);
}
```

Donanım Bağlantısı



CAN Kütüphanesi

mikroC, CAN birimleri ile çalışmayı sağlayacak bir kütüphane bulundurmaktadır.

CAN; hata yakalama, bildirme, öz-denetleme ve hata sınırlama ve özelliklerine sahip güvenilir bir sinyal iletişim protokolüdür. Hatalı CAN verileri ve çerçeveleri (remote frames) Ethernet protokolünde olduğu gibi tekrar iletilirler.

Veri transfer hızı 40 m'den kısa ağlarda 1Mbit/s'a kadar ulaşır. Ancak hız; 250 m. ağlarda 250 Kbit/s'a, daha uzun mesafelerde ise 200 Kbit/s'a düşmektedir. CAN protokolünde kullanılan kablolar ekranlı burulmuş-çiftler olup maksimum kablo uzunluğu 1000 m'dir.

CAN iki çeşit mesaj formatını destekler:
11 tanımlayıcı bit ile "Standart format", ve
29 tanımlayıcı bit ile "Genişletilmiş format" ,

Not: CAN yordamları şu anda sadece P18xxx8 PIC MCU'ları tarafından desteklenmektedir. Arabirim olarak mikro-denetleyici; "CAN çoklu veri hattına" (CAN BUS) bağlanmış MCP2551 ve benzerleri gibi CAN alıcı-vericilerine bağlanmalıdır.
Not: Bazı fonksiyonları kullanabilmek için CAN sabitlerinin gerekli olup olmadığına emin olunuz. Sayfa 170'e bakınız.

Kütüphane Yordamları

```
CANSetOperationMode
CANGetOperationMode
CANInitialize
CANSetBaudRate
CANSetMask
CANSetFilter
CANRead
CANWrite
```

Aşağıdaki yordamlar yalnızca derleyicinin iç kullanımı içindir:

```
RegsToCANID
CANIDToRegs
```

CANSetOperationMode

Yapısı	`void CANSetOperationMode(char mode, char wait_flag);`
Tanımı	CAN'ı istenen çalışma durumuna ayarlar; yani modu CANSTAT yazmacına kopyalar. `mode` parametresi CAN_OP_MODE sabitlerinden biri olmalıdır (CAN sabitlerine bakınız). `wait_flag` parametresi 0 veya 0xFF olmalıdır: Eğer 0xFF olarak ayarlı ise, bu bir tıkanan (blocking) çağırmadır; istenen durum set olmadığı müddetçe fonksiyon dönmeyecektir. Eğer 0'a ayarlı ise, bu bir tıkanmayan (non-blocking) çağırmadır dolayısıyla CAN modülün istenen duruma anahtarlandığına bakmayacaktır. Çağırıcı (caller) moda özgün işlemler yapmadan önce doğru çalışma modunun seçildiğini belirlemek için `CANGetOperationMode` fonksiyonunu kullanmalıdır.
Gereklilikler	Şu anda CAN yordamları sadece P18xxx8 PIC MCU'ları tarafından doğrudan desteklenmektedirler. Mikrodenetleyiciler; CAN çoklu-veri hattına bağlı CAN alıcı-vericilerine bağlanmalıdırlar (MCP2551 veya benzerleri).
Örnek	`CANSetOperationMode(CAN_MODE_CONFIG, 0xFF);`

CANGetOperationMode

Yapısı	`char CANGetOperationMode(void);`
Dönüş	Geçerli işlem modu (opmode).
Tanımı	Fonksiyon, CAN biriminin geçerli işlem modunu döndürür.
Gereklilikler	CAN yordamları sadece P18xxx8 PIC MCU'ları tarafından doğrudan desteklenmektedirler. Mikrodenetleyiciler; CAN çoklu veri-hattına bağlı CAN alıcı-vericilerine bağlanmalıdırlar (MCP2551 veya benzerleri).
Örnek	`if (CANGetOperationMode() == CAN_MODE_NORMAL) { ... };`

CANInitialize

Yapısı	`void CANInitialize(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`

Tanımı	CAN başlangıç değerlerini ayarlar. Tüm bekleyen iletimler iptal edilir. Tüm mesajlara izin verebilmek için tüm maske yazmaçlarını 0'a ayarlar. Bu fonksiyon konfigürasyon modunu içerden ayarlar. Fonksiyon çağrıldığında Normal mod seçilir.
	
	Filtreli yazmaçlar bayrak değerlerine göre ayarlanırlar:
	
	`if (CAN_CONFIG_FLAGS & CAN_CONFIG_VALID_XTD_MSG != 0)`
	`  // Tüm filtreleri  XTD_MSG'ye kur`
	`else if (config & CONFIG_VALID_STD_MSG != 0)`
	`  // Tüm filtreleri  STD_MSG' ye kur`
	`else`
	`  // Filtrelerin yarisini  STD' ye, geri kalanini XTD_MSG' ye kur.`
	
	Parametreler:
	
	`SJW` 18XXX8 dokümanında tanımlanmıştır (1–4)
	`BRP` 18XXX8 dokümanında tanımlanmıştır (1–64)
	`PHSEG1` 18XXX8 dokümanında tanımlanmıştır (1–8)
	`PHSEG2` 18XXX8 dokümanında tanımlanmıştır (1–8)
	`PROPSEG` 18XXX8 dokümanıda tanımlanmıştır (1–8)
	`CAN_CONFIG_FLAGS` önceden belirtilmiş sabitlerden oluşturulur.(Bkz: CAN sabitleri)

Gereklilikler	Halen CAN yordamları sadece P18xxx8 PIC MCU'ları tarafından doğrudan desteklemektedirler. Mikrodenetleyiciler; CAN çoklu-veri hattına bağlı CAN alıcı-vericilerine bağlanmalıdırlar (MCP2551 veya benzerleri).

Örnek	`init = CAN_CONFIG_SAMPLE_THRICE  &`
	`  CAN_CONFIG_PHSEG2_PRG_ON  &`
	`  CAN_CONFIG_STD_MSG  &`
	`  CAN_CONFIG_DBL_BUFFER_ON  &`
	`  CAN_CONFIG_VALID_XTD_MSG  &`
	`  CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANInitialize(1, 1, 3, 3, 1, init);  //  CAN ilk degerlerini ver`

CANSetBaudRate

Yapısı	`void CANSetBaudRate(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
Tanımı	CAN baud hızını ayarlar. CAN protokolünün karmaşıklığından dolayı, zoraki bir bps (saniyedeki bit sayısı) değerini veremezsiniz. Bunun yerine CAN Config modunda olduğu zaman bu fonksiyonu kullanınız. Detaylı bilgi için veri dokümanına bakınız. Parametreler: `SJW`; 18XXX8 dokümanı (1-4)'de tanımlandığı gibi `BRP`; 18XXX8 dokümanı (1–64)'de tanımlandığı gibi `PHSEG1`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `PHSEG2`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `PROPSEG`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `CAN_CONFIG_FLAGS` önceden tanımlanmış sabitlerden oluşturulur (CAN sabitlerine bakınız).
Gereklilikler	CAN, Config (kurulum) modunda olmalıdır. Aksi halde fonksiyon ihmal edilir.

Örnek	`init = CAN_CONFIG_SAMPLE_THRICE & CAN_CONFIG_PHSEG2_PRG_ON & CAN_CONFIG_STD_MSG & CAN_CONFIG_DBL_BUFFER_ON & CAN_CONFIG_VALID_XTD_MSG & CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANSetBaudRate(1, 1, 3, 3, 1, init);`

CANSetMask

Yapısı	`void CANSetMask(char CAN_MASK, long value, char CAN_CONFIG_FLAGS);`
Tanımı	Fonksiyon, mesajları gelişmiş olarak filtrelemek için maskeleri ayarlar. Verilen değer tampon (buffer) maske yazmacının uygun bitini ayarlar. Parametreler: `CAN_MASK` önceden tanımlanmış sabitlerden biridir (CAN sabitlerine bakınız); `value` maske yazmacının değeridir; `CAN_CONFIG_FLAGS` filtrelenecek mesaj tipini seçer; ya `CAN_CONFIG_XTD_MSG` ya da `CAN_CONFIG_STD_MSG`' den birini.
Gereklilikler	CAN, Config modunda olmalıdır, aksi halde fonksiyon ihmal edilir.

Örnek	`// Tum maske bitlerini 1'e kur, yani tum filtreli bitler uygun:`
	`CANSetMask(CAN_MASK_B1, -1, CAN_CONFIG_XTD_MSG);`
	
	`/* Not: -1 yazmak 0xFFFFFFFF yazmanin daha pratik bir yoludur. Tumleme (complement) islemi isimizi gorecek ve degerin tumunu birler ile dolduracaktır. */`

CANSetFilter

Yapısı	`void CANSetFilter(char CAN_FILTER, long value, char CAN_CONFIG_FLAGS);`
Tanımı	Fonksiyon, mesajları gelişmiş olarak filtrelemek için maskeleri ayarlar. Verilen değer tampon (buffer) maske yazmacının uygun bitini ayarlar. Parametreler: `CAN_MASK` önceden tanımlanmış sabitlerden biridir (CAN sabitlerine bakınız); `value` filtre yazmacının değeridir; `CAN_CONFIG_FLAGS` filtrelenecek mesajın tipini seçer; ya `CAN_CONFIG_XTD_MSG` ya da `CAN_CONFIG_STD_MSG`'den biri
Gereklilikler	CAN, Config modunda olmalıdır. Aksi halde fonksiyon ihmal edilir.
Örnek	`/* B1_F1'in id filtresini 3'e kur: */` `CANSetFilter(CAN_FILTER_B1_F1, 3, CAN_CONFIG_XTD_MSG);`

CANRead

Yapısı	`char CANRead(long *id, char *data, char *datalen, char *CAN_RX_MSG_FLAGS);`
Dönüş	Alıcı tamponun içeriğini döndürür. Eğer hiçbir mesaj bulunmazsa sıfır döndürür.
Tanımı	Fonksiyon alıcı tamponundan (buffer) mesaj okur. Eğer en az bir birim dolu alıcı tampon bulunursa, bu ayrıştırılır ve döndürülür. Eğer hiç dolu tampon bulunmazsa, fonksiyon sıfır döndürür. Parametreler : `id` mesaj tanıtıcısıdır (identifier);. `data` 8 byte'a kadar uzayabilen baytlar dizisidir. `datalen`; 1-8 byte arasında veri uzunluğudur. `CAN_TX_MSG_FLAGS` sabitlerden düzenlenmiş değerdir (CAN sabitlerine bakınız).
Gereklilikler	CAN, veri alımına uygun bir modda olmalıdır.
Örnek	`char rcv, rx, len, data[8]; long id;` `rcv = CANRead(id, data, len, 0);`

CANWrite

Yapısı	`char CANWrite(long id, char *data, char datalen, char CAN_TX_MSG_FLAGS);`
Dönüş	Eğer mesaj kuyruğa konamazsa; yani tampon doluysa sıfır döndürür.
Tanımı	Eğer en az bir boş gönderici tamponu bulunursa fonksiyon mesajı gönderim için kuyruğa (queue) yollar. Eğer tampon dolu ise, fonksiyon sıfır döndürür. Parametreler:`id`; CAN mesaj tanıtıcısıdır. Mesaj tipine bağlı olarak sadece 11 veya 29 bit kullanılabilir (standart veya genişletilmiş). `data` 8 byte'a kadar uzayabilen baytlar dizisidir. `datalen`; 1-8 byte arasında veri uzunluğudur. `CAN_TX_MSG_FLAGS` sabitlerden düzenlenmiş değerdir (CAN sabitlerine bakınız).
Gereklilikler	CAN, normal modda olmalıdır.

Örnek	`char tx, data; long id;`
	`tx = CAN_TX_PRIORITY_0 & CAN_TX_XTD_FRAME;`
	`CANWrite(id, data, 2, tx);`

CAN Sabitleri

CAN kütüphanesinde tanımlanmış birçok sabit bulunmaktadır. Kütüphaneleri etkin bir şekilde kullanmak için onlara aşina olmanız gerekmektedir. Bölüm sonundaki örneği inceleyebilirsiniz.

CAN_OP_MODE

CAN_OP_MODE sabitleri, CAN işlem modunu tanımlar.
CAN_Set_Operation_Mode fonksiyonu aşağdakilerden birini gerçek bağımsız değişken olarak bekler:

```
#define CAN_MODE_BITS      0xE0   // mod bit'lere erismek icin kullan
#define CAN_MODE_NORMAL    0
#define CAN_MODE_SLEEP     0x20
#define CAN_MODE_LOOP      0x40
#define CAN_MODE_LISTEN    0x60
#define CAN_MODE_CONFIG    0x80
```

CAN_CONFIG_FLAGS

CAN_CONFIG_FLAGS sabitleri, CAN biriminin ayarları ile ilgili bayrakları belirler. CANInitialize ve CANSetBaudRate fonksiyonları aşağıdaki değişkenlerden birini (veya bitsel bileşimlerini) bekler:

```
#define CAN_CONFIG_DEFAULT              0xFF   // 11111111

#define CAN_CONFIG_PHSEG2_PRG_BIT       0x01
#define CAN_CONFIG_PHSEG2_PRG_ON        0xFF   // XXXXXXX1
#define CAN_CONFIG_PHSEG2_PRG_OFF       0xFE   // XXXXXXX0

#define CAN_CONFIG_LINE_FILTER_BIT      0x02
#define CAN_CONFIG_LINE_FILTER_ON       0xFF   // XXXXXX1X
#define CAN_CONFIG_LINE_FILTER_OFF      0xFD   // XXXXXX0X

#define CAN_CONFIG_SAMPLE_BIT           0x04
#define CAN_CONFIG_SAMPLE_ONCE          0xFF   // XXXXX1XX
#define CAN_CONFIG_SAMPLE_THRICE        0xFB   // XXXXX0XX

#define CAN_CONFIG_MSG_TYPE_BIT         0x08
#define CAN_CONFIG_STD_MSG              0xFF   // XXXX1XXX
#define CAN_CONFIG_XTD_MSG              0xF7   // XXXX0XXX

// devam ediyor..
```

```
// ..devam

#define CAN_CONFIG_DBL_BUFFER_BIT       0x10
#define CAN_CONFIG_DBL_BUFFER_ON        0xFF    // XXX1XXXX
#define CAN_CONFIG_DBL_BUFFER_OFF       0xEF    // XXX0XXXX

#define CAN_CONFIG_MSG_BITS             0x60
#define CAN_CONFIG_ALL_MSG              0xFF    // X11XXXXX
#define CAN_CONFIG_VALID_XTD_MSG        0xDF    // X10XXXXX
#define CAN_CONFIG_VALID_STD_MSG        0xBF    // X01XXXXX
#define CAN_CONFIG_ALL_VALID_MSG        0x9F    // X00XXXXX
```

Bu değerlerden config byte'ı ortaya koyabilmeniz için bitsel AND (&) işlemini uygulamalısınız. Örneğin:

```
init =  CAN_CONFIG_SAMPLE_THRICE  & CAN_CONFIG_PHSEG2_PRG_ON &
        CAN_CONFIG_STD_MSG        & CAN_CONFIG_DBL_BUFFER_ON &
        CAN_CONFIG_VALID_XTD_MSG  & CAN_CONFIG_LINE_FILTER_OFF;
//...
CANInitialize(1, 1, 3, 3, 1, init);   // CAN'i baslamaya hazirla
```

CAN_TX_MSG_FLAGS

CAN_TX_MSG_FLAGS, bir CAN mesajının iletilmesi ile ilgili bayraklardır:

```
#define CAN_TX_PRIORITY_BITS      0x03
#define CAN_TX_PRIORITY_0         0xFC  // XXXXXX00
#define CAN_TX_PRIORITY_1         0xFD  // XXXXXX01
#define CAN_TX_PRIORITY_2         0xFE  // XXXXXX10
#define CAN_TX_PRIORITY_3         0xFF  // XXXXXX11

#define CAN_TX_FRAME_BIT          0x08
#define CAN_TX_STD_FRAME          0xFF  // XXXXX1XX
#define CAN_TX_XTD_FRAME          0xF7  // XXXXX0XX

#define CAN_TX_RTR_BIT            0x40
#define CAN_TX_NO_RTR_FRAME       0xFF  // X1XXXXXX
#define CAN_TX_RTR_FRAME          0xBF  // X0XXXXXX
```

Uygun bayrakları ayarlamak için bitsel AND (&) işlemini uygulamalısınız. Örneğin:

```
/* CANSendMessage ile kullanilacak degeri hazirlayalim: */
send_config = CAN_TX_PRIORITY_0 && CAN_TX_XTD_FRAME &
              CAN_TX_NO_RTR_FRAME;
//...
CANSendMessage(id, data, 1, send_config);
```

CAN_RX_MSG_FLAGS

CAN_RX_MSG_FLAGS bir CAN mesajının alımı ile ilgili bayraklardır. Eğer özel bir bit set olursa, karşılık gelen anlam geçerli (TRUE), aksi halde geçersiz (FALSE)'dir.

```
#define CAN_RX_FILTER_BITS  0x07  // Filtre bitlerine erisim icin
#define CAN_RX_FILTER_1     0x00
#define CAN_RX_FILTER_2     0x01
#define CAN_RX_FILTER_3     0x02
#define CAN_RX_FILTER_4     0x03
#define CAN_RX_FILTER_5     0x04
#define CAN_RX_FILTER_6     0x05
#define CAN_RX_OVERFLOW     0x08  // Tasma varsa 1; yoksa 0
#define CAN_RX_INVALID_MSG  0x10  // Gecersizse 1; yoksa 0
#define CAN_RX_XTD_FRAME    0x20  // XTD msg ise 1; yoksa 0
#define CAN_RX_RTR_FRAME    0x40  // RTR msg ise 1; yoksa 0
#define CAN_RX_DBL_BUFFERED 0x80  // Eger msg donanimsal olarak
                                  // cift tampon bellekliyse
                                  // (double-buffered)
```

Uygun bayrakları ayarlamak için bitsel AND (&) işlemini uygulamalısınız. Örneğin:

```
if (MsgFlag & CAN_RX_OVERFLOW != 0) {
  ... // Alici kapasitesi asildi (Tasma durumu)
      // ve onceki mesaj kaybedildi.
}
```

CAN_MASK

CAN_MASK sabitleri maske kodlarını tanımlamaktadırlar. CANSetMask fonksiyonu aşağıdakilerden birini gerçek bağımsız değişkeni olarak bekler:

```
#define CAN_MASK_B1  0
#define CAN_MASK_B2  1
```

CAN_FILTER

CAN_FILTER sabitleri filtre kodlarını içermektedir. CANSetFilter fonksiyonu aşağıdakilerden birini gerçek bağımsız değişkeni olarak bekler:

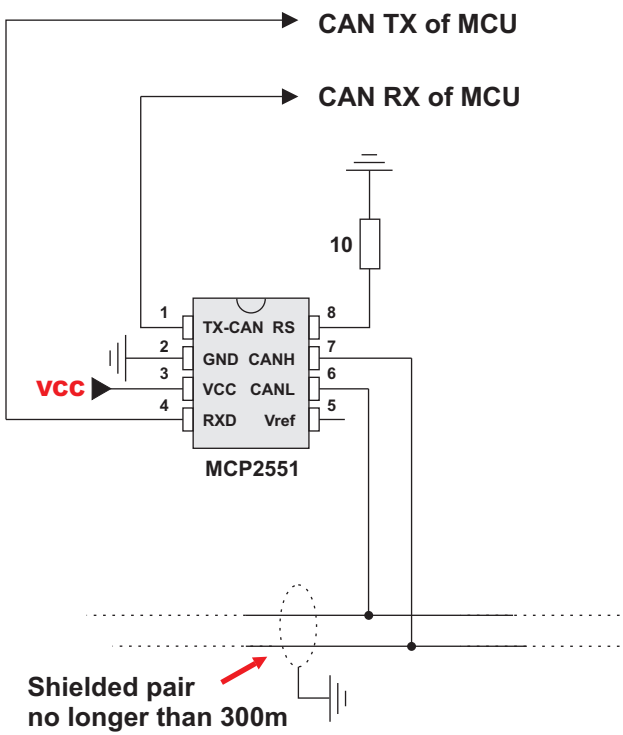
```
#define CAN_FILTER_B1_F1  0
#define CAN_FILTER_B1_F2  1
#define CAN_FILTER_B2_F1  2
#define CAN_FILTER_B2_F2  3
#define CAN_FILTER_B2_F3  4
#define CAN_FILTER_B2_F4  5
```

Kütüphane Örneği

```
unsigned short aa, aa1, len, aa2;
unsigned char data[ 8] ;
long id;
unsigned short zr, cont, oldstate;

//........

void main() {
  PORTC = 0;
  TRISC = 0;
  PORTD = 0;
  TRISD = 0;
  aa  = 0;
  aa1 = 0;
  aa2 = 0;

  // CANSendMessage ile kullanilacak degeri hazirlayalim:
  aa1 =  CAN_TX_PRIORITY_0 &
         CAN_TX_XTD_FRAME &
         CAN_TX_NO_RTR_FRAME;

  // CANInitialize ile kullanilacak degeri hazirlayalim:
  aa =   CAN_CONFIG_SAMPLE_THRICE      &
         CAN_CONFIG_PHSEG2_PRG_ON      &
         CAN_CONFIG_STD_MSG            &
         CAN_CONFIG_DBL_BUFFER_ON      &
         CAN_CONFIG_VALID_XTD_MSG      &
         CAN_CONFIG_LINE_FILTER_OFF;

  data[ 0]  = 0;

  // CAN'i baslangic durumuna getirelim
  CANInitialize(1,1,3,3,1,aa);

  // CAN'i CONFIG modunda kuralim
  CANSetOperationMode(CAN_MODE_CONFIG,0xFF);

  id = -1;


// devam ediyor..
```

```
// .. devam

  // Tum mask1 bitlerini 1 yap
  CANSetMask(CAN_MASK_B1,ID,CAN_CONFIG_XTD_MSG);

  // Tum mask2 bitlerini 1 yap
  CANSetMask(CAN_MASK_B2,ID,CAN_CONFIG_XTD_MSG);

  // B1_F1 filtresini 3'e kur
  CANSetFilter(CAN_FILTER_B2_F3,3,CAN_CONFIG_XTD_MSG);

  // CAN'i  NORMAL duruma kur
  CANSetOperationMode(CAN_MODE_NORMAL,0xFF);

  PORTD = 0xFF;
  id = 12111;
  CANWrite(id,data,1,aa1);        // CAN  yoluyla mesaj yolla

  while (1) {
    oldstate = 0;
    zr = CANRead(&id, data , &len, &aa2);
    if ((id == 3) & zr) {
      PORTD = 0xAA;
      PORTC = data[0];           // Veriyi PORTC'ye yaz
      data[0]++ ;

      /* Eger mesajda iki veri bayti varsa, ikincisini PORTD'ye yaz */
      if (len == 2)  PORTD = data[1];

      data[1]  = 0xFF;
      id = 12111;
      CANWrite(id, data, 2,aa1); // Arttirilan veriyi geri gonder
    }
  }
}//~!
```

Donanım Bağlantısı



CANSPI Kütüphanesi

SPI (Serial Peripheral Interface - Seri Çevresel Donanım Arabirimi) birimi bazı PIC MCU'larda mevcuttur. mikroC, MCP2515 veya MCP2510 gibi dış CAN birimlerinin SPI yoluyla çalışmasını sağlamak amacıyla bir kütüphane (sürücü-driver) sağlar.

mikroC'deki CAN kütüphanesinin her yordamının aynı söz dizimine sahip CANSPI karşılığı vardır. Denetleyici Alan Ağı (Controller Area Network-CAN) ile ilgili daha fazla bilgi için, CAN kütüphanelerine bakınız. Burada etkin haberleşme hızı SPI'ya bağlı olup "gerçek" CAN'dan şüphesiz daha yavaştır.

Not: CANSPI fonksiyonları PORTC üzerinde SPI ya sahip olan her PIC MCU tarafından sağlanır. Ayrıca MCP2510 veya MCP2515'in CS (chip-select) pini RC0'a bağlı olmalıdır. Örnek devrenin donanım yapısı bölüm sonunda verilmiştir.

Not: CANSPI'yı başlatılmadan önce `SPI_Init()` yordamı çağrılmalıdır. Sayfa 325'e bakınız.

Kütüphane Yordamları

```
CANSPISetOperationMode
CANSPIGetOperationMode
CANSPIInitialize
CANSPISetBaudRate
CANSPISetMask
CANSPISetFilter
CANSPIRead
CANSPIWrite
```

Aşağıdaki yordamlar yalnızca derleyicinin iç kullanımı içindir:

```
RegsToCANSPIID
CANSPIIDToRegs
```

CANSPISetOperationMode

Yapısı	`void CANSPISetOperationMode(char mode, char wait_flag);`
Tanımı	CAN'ı, istenen moda ayarlar, yani modu CANSTAT'a kopyalar. `mode` parametresi `CAN_OP_MODE` sabitlerinden biri olmalıdır (CAN sabitlerine bakınız; Sayfa 170). `wait_flag` parametresi 0 veya 0xFF olmalıdır: Eğer 0xFF'e ayarlı ise, bu tıkanan (blocking) bir çağrıdır; istenen mod ayarlanana kadar fonksiyon geri dönmeyecektir. Eğer 0'a ayarlı ise, bu tıkanmayan bir çağrıdır. CAN biriminin istenen moda ayarlanıp ayarlanmadığına bakmayacaktır. Çağırıcı (caller) moda özgün işlemler yapmadan önce doğru çalışma modunun seçildiğini belirlemek için `CANSPIGetOperationMode` fonksiyonunu kullanmalıdır.
Gereklilikler	CANSPI fonksiyonları PORTC'de SPI arabirimine sahip herhangi bir PIC MCU tarafından desteklenir. Ayrıca MCP2510 veya MCP2515'in CS (chip-select) pini RC0'a bağlanmalıdır.
Örnek	`CANSPISetOperationMode(CAN_MODE_CONFIG, 0xFF);`

CANSPIGetOperationMode

Yapısı	`char CANSPIGetOperationMode(void);`
Dönüş	Geçerli olan işlem modu.
Tanımı	Fonksiyon CAN biriminin geçerli olan işlem modunu döndürür.
Örnek	`if (CANSPIGetOperationMode() == CAN_MODE_NORMAL) { ... };`

CANSPIInitialize

Yapısı	`void CANSPIInitialize(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS, char * RstPort, char RstPin, char * CSPort, char CSPin);`

Tanımı	CANSPI 'ın başlangıç ayarlarını yapar. Tüm bekleyen iletimler iptal edilir. Tüm maske yazmaçları 0 yapılarak tüm mesajların alımına izin verilir.
	
	Filtre yazmaçları aşağıdaki bayrak değerlerine göre ayarlanırlar:
	
	`if (CAN_CONFIG_FLAGS & CAN_CONFIG_VALID_XTD_MSG != 0)`
	`  // Tüm filtreleri  XTD_MSG'ye ayarlar.`
	`else if (config & CONFIG_VALID_STD_MSG != 0)`
	`  // Tüm filtreleri  STD_MSG'ye ayarlar.`
	`else`
	`  // Filtrelerin yarisini STD'ye, kalanini XTD_MSG'ye ayarlar.`
	
	Parametreler:
	`SJW`; 18XXX8 dokümanı (1-4)'de tanımlandığı gibi
	`BRP`; 18XXX8 dokümanı (1–64)'de tanımlandığı gibi
	`PHSEG1`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi
	`PHSEG2`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi
	`PROPSEG`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi
	
	`CAN_CONFIG_FLAGS` önceden tanımlanmış sabitlerden oluşturulur (CAN sabitlerine bakınız; Sayfa 170).

Gereklilikler	CANSPI başlatılmadan önce `SPI_Init()` çağırılmalıdır. CANSPI, Config durumda olmalıdır; aksi halde fonksiyon ihmal edilecektir.

Örnek	`init = CAN_CONFIG_SAMPLE_THRICE  &`
	`  CAN_CONFIG_PHSEG2_PRG_ON  &`
	`  CAN_CONFIG_STD_MSG  &`
	`  CAN_CONFIG_DBL_BUFFER_ON  &`
	`  CAN_CONFIG_VALID_XTD_MSG  &`
	`  CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`// Harici CAN modulunun baslangic ayarlarini`
	`// SPI uzerinden yap.`
	
	`CANSPIInitialize( 1,1,3,3,1,init, &PORTC, 2, &PORTC, 0);`

CANSPISetBaudRate

Yapısı	`void CANSPISetBaudRate(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
Tanımı	CANSPI baud hızını ayarlar. CANSPI protokolünün karmaşıklığından dolayı, zoraki bir bps (saniyedeki bit sayısı) değerini veremezsiniz. Bunun yerine, CAN Config modunda olduğu zaman bu fonksiyonu kullanınız. Detaylı bilgi için dökümanına bakınız. Parametreler: `SJW`; 18XXX8 dokümanı (1-4)' de tanımlandığı gibi `BRP`; 18XXX8 dokümanı (1–64)'de tanımlandığı gibi `PHSEG1`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `PHSEG2`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `PROPSEG`; 18XXX8 dokümanı (1–8)'de tanımlandığı gibi `CAN_CONFIG_FLAGS` önceden tanımlanmış sabitlerden oluşturulur (CAN sabitlerine bakınız; Sayfa 170).
Gereklilikler	CANSPI, Config (kurulum) modunda olmalıdır. Aksi halde fonksiyon ihmal edilir.

Örnek	`init = CAN_CONFIG_SAMPLE_THRICE & CAN_CONFIG_PHSEG2_PRG_ON & CAN_CONFIG_STD_MSG & CAN_CONFIG_DBL_BUFFER_ON & CAN_CONFIG_VALID_XTD_MSG & CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANSPISetBaudRate(1, 1, 3, 3, 1, init);`

CANSPISetMask

Yapısı	`void CANSPISetMask(char CAN_MASK, long value, char CAN_CONFIG_FLAGS);`
Tanımı	Fonksiyon, mesajları yüksek seviyede filtrelemek için maskeleri ayarlar. Verilen değer tampon (buffer) maske kayıtçısının uygun bitini ayarlar. Parametreler: `CAN_MASK` önceden tanımlanmış sabitlerdendir (CAN sabitlerine bakınız); `value` maske kayıtçısının değeridir; `CAN_CONFIG_FLAGS` filtrelenecek mesaj tipini seçer, ya `CAN_CONFIG_XTD_MSG` ya da `CAN_CONFIG_STD_MSG`'den birini.
Gereklilikler	CANSPI, Config modunda olmalıdır; aksi halde fonksiyon ihmal edilecektir.

Örnek	`// Tum maske bitlerini 1'e kur, yani tum filtreli bitler anlamli:`
	`CANSPISetMask(CAN_MASK_B1, -1, CAN_CONFIG_XTD_MSG);`
	
	`/* Not: -1 yazmak 0xFFFFFFFF yazmanin daha pratik bir yoludur. Tumleme (complement) islemi isimizi gorecek ve degerin tumunu bir-ler ile dolduracaktir. */`

CANSPISetFilter

Yapısı	`void CANSPISetFilter(char CAN_FILTER, long value, char CAN_CONFIG_FLAGS);`
Tanımı	Fonksiyon, mesajları gelişmiş olarak filtrelemek için maskeleri ayarlar. Verilen değer tampon (buffer) maske yazmacının uygun bitini ayarlar. Parametreler: `CAN_MASK` önceden tanımlanmış sabitlerdendir (CAN sabitlerine bakınız); `value` maske yazmacının değeridir; `CAN_CONFIG_FLAGS` filtrelenecek mesaj tipini seçer, ya `CAN_CONFIG_XTD_MSG` ya da `CAN_CONFIG_STD_MSG`' den birini.
Gereklilikler	CANSPI, Config modunda olmalıdır; aksi halde fonksiyon ihmal edilecektir.
Örnek	`/* B1_F1 filtresinin ID'sini 3'e kur:` `CANSPISetFilter(CAN_FILTER_B1_F1, 3, CAN_CONFIG_XTD_MSG);`

CANSPIRead

Yapısı	`char CANSPIRead(long *id, char *data, char *datalen, char *CAN_RX_MSG_FLAGS);`
Dönüş	Mesaj gelmişse, alış tamponundaki mesajı döndürür. Eğer hiç bir mesaj gelmemişse sıfır döndürür.
Tanımı	Fonksiyon alış tamponundan (buffer) mesaj okur. Eğer sonunda en az bir birim dolu alıcı tampon bulunursa, bu ayrıştırılır ve döndürülür. Eğer hiçbir tampon bulunmazsa fonksiyon sıfır döndürür. Parametreler : `id` mesaj tanıtıcısıdır (identifier). `data` 8 byte'a kadar uzayabilen baytlar dizisidir. `datalen` 1-8 bayt arasında veri uzunluğudur; `CAN_RX_MSG_FLAGS` sabitlerden oluşturulmuş değerdir (Bakınız CAN sabitleri).
Gereklilikler	CANSPI, veri alımına uygun bir modda olmalıdır.
Örnek	`char rcv, rx, len, data[8]; long id;` `rcv = CANSPIRead(id, data, len, 0);`

CANSPIWrite

Yapısı	`char CANSPIWrite(long id, char *data, char datalen, char CAN_TX_MSG_FLAGS);`
Dönüş	Eğer mesaj kuyruğa konamazsa; yani tampon doluysa sıfır döndürür.
Tanımı	Eğer en az bir boş gönderici tamponu bulunursa fonksiyon mesajı gönderim için kuyruğa (queue) yollar. Eğer tampon dolu ise, fonksiyon sıfır döndürür. Parametreler: `id`; CANSPI mesaj tanıtıcısıdır. Mesaj tipine bağlı olarak sadece 11 veya 29 bit kullanılabilir (standart veya genişletilmiş). `data` 8 byte'a kadar uzayabilen byte'lar dizisidir. `datalen`; 1-8 byte arasında veri uzunluğudur. `CAN_TX_MSG_FLAGS` sabitlerden düzenlenmiş değerdir .
Gereklilikler	CANSPI, normal modda olmalıdır.

Örnek	`char tx, data; long id;`
	`tx = CAN_TX_PRIORITY_0 & CAN_TX_XTD_FRAME;`
	`CANSPIWrite(id, data, 2, tx);`

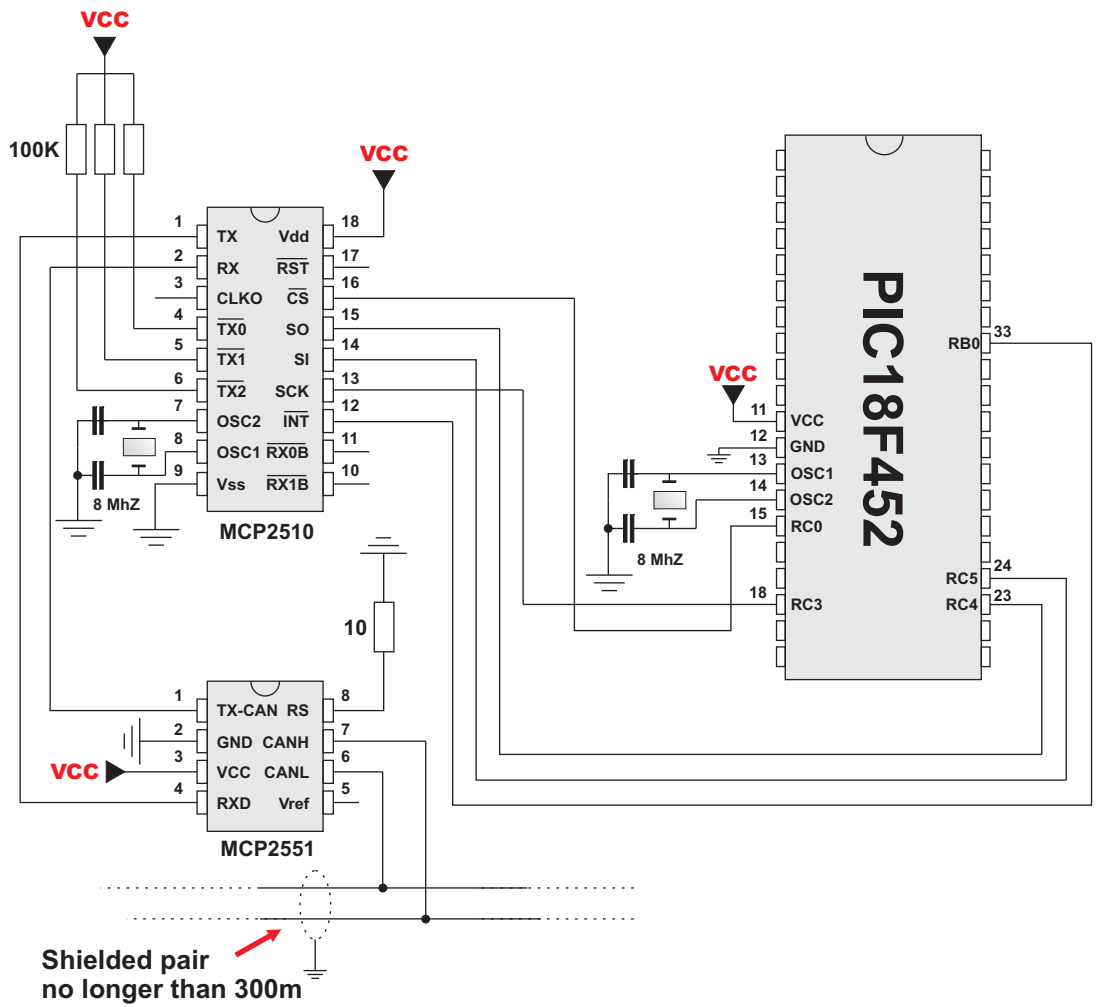
Kütüphane örneği

Aşağıdaki program CANSPI protokolünü tanıtan bir örnektir. İki PIC mikrodenetleyici ünitesi (MCU) arasında basit bir veri alışverişini göstermektedir. Burada sayısal veri her karşılıklı göndermede bir artmaktadır. Veri iletişimini gözle izlemek için verinin alt byte'ı PORTC'ye, üst byte'ı ise PORTD'ye yazdırılmaktadır.

```
char data[ 8],aa, aa1, len, aa2;
long id;
char zr;
const char _TRUE  = 0xFF;
const char _FALSE = 0x00;

void main(){
  TRISB = 0;
  Spi_Init();       // SPI modulunun baslangic ayarlarini yap
  TRISC.F2 = 0;     // (TRISC,2)'yi temizle
  PORTC.F2 = 0;     // (PORTC,2)'yi temizle
  PORTC.F0 = 1;     // (PORTC,0)'yi kur
  TRISC.F0 = 0;     // (TRISC,0)'yi temizle
  PORTD = 0;
  TRISD = 0;
  aa    = 0;
  aa1   = 0;
  aa2   = 0;


  //  CANSPIInitialize ile kullanilacak degeri olustur
  aa = CAN_CONFIG_SAMPLE_THRICE  &
       CAN_CONFIG_PHSEG2_PRG_ON  &
       CAN_CONFIG_STD_MSG        &
       CAN_CONFIG_DBL_BUFFER_ON  &
       CAN_CONFIG_VALID_XTD_MSG;

  PORTC.F2 = 1;  // (PORTC,2)'yi kur

  // CANSPISendMessage ile kullanilacak degeri olustur
  aa1 = CAN_TX_PRIORITY_0 &
        CAN_TX_XTD_FRAME &
        CAN_TX_NO_RTR_FRAME;

  PORTC.F0 = 1;  // (PORTC,0)'yi kur


// devam ediyor..
```

```
// .. devam

  Spi_Init(); //  SPI baslangic ayarlarini yap

  // Harici CAN modulunun baslangic ayarlarini yap
  CANSPIInitialize( 1,1,3,3,1,aa, &PORTC, 2, &PORTC, 0);

  //  CANSPI'ı CONFIG moda ayarla
  CANSPISetOperationMode(CAN_MODE_CONFIG,_TRUE);
  ID = -1;

  // Tum mask1 bitlerini 1'e kur
  CANSPISetMask(CAN_MASK_B1,id,CAN_CONFIG_XTD_MSG);

  // Tum mask2 bitlerini 1'e kur
  CANSPISetMask(CAN_MASK_B2,id,CAN_CONFIG_XTD_MSG);

  // B1_F1 filtresini 12111'e kur
  CANSPISetFilter(CAN_FILTER_B2_F4,12111,CAN_CONFIG_XTD_MSG);

  // CANSPI mormal moda ayarla
  CANSPISetOperationMode(CAN_MODE_NORMAL,_TRUE);

  while (1) {
    zr = CANSPIRead(&id , &Data , &len, &aa2);      // Eger varsa veri al
    if (id == 12111 & zr )  {
      PORTB = data[ 0] ++ ;                         // Veriyi PORTB'ye yaz
      id = 3;
      Delay_ms(500);

      // Arttirilan veriyi geri gonder
      CANSPIWrite(id,&data,1,aa1);

      // Eger mesaj iki veri bayti iceriyorsa, ikinci bayti PORTD'ye yaz
      if (len == 2) PORTD = data[ 1] ;
    }
  }
} //~!
```

Donanım Bağlantısı



Compact Flash Kütüphanesi

Compact Flash kütüphanesi, Compact Flash kart üzerindeki veriye erişimi sağlar. (metnin devamında CF kısaltması kullanılacaktır). CF kartlar geniş bir kullanım alanına sahip bellek elemanlarıdır. Genelde dijital kameralarda kullanılırlar. Büyük kapasiteleri(8Mbyte-2 Gbyte ve daha fazla), birkaç mikro saniyelik veri erişim süreleri, bu elemanları mikrodenetleyici uygulamaları için cazip hale getirmiştir.

CF kartlarda, veri sektörlere bölünmüştür. 1 sektör genelde 512 byte'dır (bazı eski modellerde 256 byte). Veri okuma ve yazma işlemleri doğrudan yapılmaz. 512 byte'lık tampon bellek (buffer) üzerinden yapılır. Aşağıdaki yordamlar FAT16 ve FAT32 dosya sistemli CF'ler için kullanılabilir. **NOT:** Dosya işleme yordamları sadece FAT16 dosya sistemli CF'ler ile kullanılabilir.

Önemli ! Yazma işleminden önce, boot ve FAT sektörlerine yazmadığınızdan emin olunuz, aksi halde kartınız dijital kamerada veya PC'de okunamaz olur. Winhex gibi sürücü bellek organizasyonu yazılımları; bu konuda size yardımcı olabilir.

Kütüphane Yordamları

```
Cf_Init
Cf_Detect
Cf_Total_Size
Cf_Enable
Cf_Disable
Cf_Read_Init
Cf_Read_Byte
Cf_Write_Init
Cf_Write_Byte

Cf_Fat_Init
Cf_Fat_Assign
Cf_Fat_Reset
Cf_Fat_Read
Cf_Fat_Rewrite
Cf_Fat_Append
Cf_Fat_Delete
Cf_Fat_Write
Cf_Fat_Set_File_Date
Cf_Fat_Get_File_Date
Cf_Fat_Get_File_Size
```

`CF_Set_Reg_Adr` fonksiyonu sadece derleyicinin iç kullanımı içindir.

Cf_Init

Yapısı	`void Cf_Init(char *ctrlport, char *dataport);`
Tanımı	Portları CF kart ile haberleşmeyi sağlayacak şekilde başlangıç değerlerine getirir. Parametrelerde iki farklı port belirtilir: ctrlport ve dataport. `ctrlport` kontrol portudur, `dataport` veri portudur.
Örnek	`Cf_Init(&PORTB, &PORTD);`

Cf_Detect

Yapısı	`char Cf_Detect(void);`
Dönüş	CF takılıysa 1 döndürür, değilse 0 döndürür..
Tanımı	CF kartının `ctrlport` üzerinde olup olmadığını kontrol eder.
Örnek	`// CF kartinin takilmasina kadar bekle:` `do nop; while (Cf_Detect() == 0);`

Cf_Total_Size

Yapısı	`unsigned long Cf_Total_Size(void);`
Dönüş	KiloByte cinsinden toplam kart boyutu.
Tanımı	KiloByte cinsinden toplam kart boyutunu döndürür.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`size = Cf_Total_Size();`

Cf_Enable

Yapısı	`void Cf_Enable(void);`
Tanımı	Aygıtı etkinleştirir. Bu yordam, sadece aygıt `Cf_Disable` kullanılarak etkisiz hale getirilmiş ise çağırılmalıdır. Bu iki yordam birlikte, çoklu aygıt ile çalışırken veri hatlarını tutmanızı/serbest bırakmanızı sağlar. Bölüm sonundaki örneğe bakınız.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`Cf_Enable();`

Cf_Disable

Yapısı	`void Cf_Disable(void);`
Tanımı	Yordam, aygıtı etkinsizleştirir ve veri hattını diğer aygıtlar için boşaltır. Aygıtı tekrar etkin hale getirmek için `Cf_Enable` çağırılmalıdır. Bu iki yordam birlikte, çoklu aygıt ile çalışırken veri hatlarını tutmanızı/serbest bırakmanızı sağlar. Bölüm sonundaki örneğe bakınız.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`Cf_Disable();`

Cf_Read_Init

Yapısı	`void Cf_Read_Init(long address, char sectcnt);`
Tanımı	CF kartını okumak için başlangıç durumuna getirir. Parametreler: `address` verinin nereden okunacağını (sektör adresini) gösterir ve `sectcnt` okumaya hazırlanan toplam bölüm sayısıdır.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`Cf_Read_Init(590, 1);`

Cf_Read_Byte

Yapısı	`char Cf_Read_Byte(void);`
Dönüş	CF'den bir bayt döndürür.
Tanımı	CF'den bir bayt okur .
Gereklilikler	CF, okuma operasyonu için başlangıç durumuna getirilmelidir. `Cf_Read_Init` yordamına bakınız.
Örnek	`PORTC = Cf_Read_Byte(); // Bayti okur ve onu PORTC'de gosterir`

Cf_Write_Init

Yapısı	`void Cf_Write_Init(long address, char sectcnt);`
Tanımı	CF kartını yazmak için başlangıç durumuna getirir. Parametreler: `address` verinin nereye yazılacağını (sektör adresini) gösterir ve `sectcnt` yazmaya hazır hale getirilmiş toplam bölüm sayısıdır.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır . `Cf_Init()` yordamına bakınız.
Örnek	`Cf_Write_Init(590, 1);`

Cf_Write_Byte

Yapısı	`void Cf_Write_Byte(char data);`
Tanımı	Bir byte veriyi CF'a yazar. Tüm 512 byte tampon belleğe transfer edilir.
Gereklilikler	CF, yazma operasyonu için başlangıç durumuna getirilmelidir. `Cf_Init()` yordamına bakınız.
Örnek	`Cf_Write_Byte(100);`

Cf_Fat_Init

Yapısı	`void Cf_Fat_Init(unsigned short *control_port, unsigned short wr, rd, a2, a1, a0, ry, ce, cd, unsigned short *data_port);`
Dönüş	Başlatma işlemi sorunsuz gerçekleşmişse 0 döndürür. Ancak başlangıç sektörü bulunamamışsa 1, kart algılanamamışsa 255 döndürür.
Tanımı	CF kartı ile birlikte FAT operasyonları için portları başlangıç değerine getirir. İki farklı port belirler: ctrlport ve dataport. wr, rd, a2, a1, a0, ry, ce ve cd kontrol portu üzerindeki pin isimleridir.
Gereklilikler	Yoktur.
Örnek	`CF_Fat_Init(PORTD,6,5,2,1,0,7,3,4, PORTC);`

Cf_Fat_Assign

Yapısı	`unsigned short Cf_Fat_Assign(char *filename, char create_file);`
Dönüş	Dosya mevcut ise (veya mevcut değil ama yeni oluşuruluyor ise) 1 döndürür. Dosya mevcut değil ve yeni dosya oluşturulmuyorsa 0 döndürür.
Tanımı	FAT operasyonları için dosya tahsis eder. Eğer dosya mevcut değilse, fonksiyon yeni bir dosya oluşturacaktır; `filename` parametresi dosyanın ismidir. (Dosya ismi 8.3 BÜYÜK HARF formatında olmalıdır.) `create_file` yeni dosya oluşturma için bir parametredir. Eğer `create_file` 0'dan farklı ise yeni bir dosya oluşturulur.
Gereklilikler	Portlar CF ile yapılacak FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız.
Örnek	`Cf_Fat_Assign('MIKROELE.TXT',1);`

Cf_Fat_Reset

Yapısı	`void Cf_Fat_Reset(unsigned long *size);`
Dönüş	Byte cinsinden dosya boyutudur. Boyut `size` giriş değişkeninin adresi üzerinde depolanır.
Tanımı	Belirlenen dosyayı okumak için açar.
Gereklilikler	Portlar CF ile yapılacak FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya atanmış olmalıdır. `Cf_Fat_Assign`'a bakınız.
Örnek	`Cf_Fat_Reset(size);`

Cf_Fat_Read

Yapısı	`void Cf_Fat_Read(unsigned short *bdata);`
Tanımı	Veriyi dosyadan okur. `bdata` dosyadan okunan veridir.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumun getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız. Dosya okumak için açılmalıdır. `Cf_Fat_Reset`'e bakınız.
Örnek	`Cf_Fat_Read(character);`

Cf_Fat_Rewrite

Yapısı	`void Cf_Fat_Rewrite();`
Dönüş	Hiçbirşey
Tanımı	Tahsis edilen dosyayı tekrardan yazar.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init'` e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız.
Örnek	`Cf_Fat_Rewrite;`

Cf_Fat_Append

Yapısı	`void Cf_fat_Append();`
Dönüş	Hiçbirşey.
Tanımı	Dosyayı yazmak için açar. Bu yordam dosyanın son byte'ından yazmaya devam eder
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edimelidir. `Cf_Fat_Assign`'a bakınız.
Örnek	`Cf_Fat_Append;`

Cf_Fat_Delete

Yapısı	`void Cf_Fat_Delete();`
Tanımı	Dosyayı CF kart'tan siler.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız.
Örnek	`Cf_Fat_Delete;`

Cf_Fat_Write

Yapısı	`void Cf_Fat_Write(char *fdata, unsigned data_len);`
Dönüş	Hiçbir şey
Tanımı	CF'ye veriyi yazar. `fdata` parameteresi CF'ye yazılan veridir. `data_len` CF'ye yazılan byte'ların sayısıdır.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız. Dosya yazma için açık olmalıdır. `Cf_Fat_Rewrite` veya `Cf_Fat_Attend`' e bakınız.
Örnek	`Cf_Fat_Write(file_contents, 42); // Secili dosyaya veriyi yaz`

Cf_Fat_Set_File_Date

Yapısı	`void Cf_fat_Set_File_Date(unsigned int year, unsigned short month, unsigned short day, unsigned short hours, unsigned short mins, unsigned short seconds);`
Dönüş	Hiçbirşey
Tanımı	Dosyanın zaman özelliklerini ayarlar. Dosyanın; yıl, ay, gün, saat, dakika ve saniyesini ayarlayabilirsiniz.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız. Dosya yazma için açık olmalıdır. `Cf_Fat_Rewrite` veya `Cf_Fat_Attend`' e bakınız.
Örnek	`Cf_Fat_Set_File_Date(2005,9,30,17,41,0);`

Cf_Fat_Get_File_Date

Yapısı	`void Cf_Fat_Get_File_Date(unsigned int *year, unsigned short *month, unsigned short *day, unsigned short *hours, unsigned short *mins);`
Tanımı	Dosyanın zaman parametrelerini okur. Dosyanın; yıl, ay, gün , saat ve dakikasını okuyabilirsiniz.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`' a bakınız.
Örnek	`Cf_Fat_Get_File_Date(year, month, day, hours, mins);`

Cf_Fat_Get_File_Size

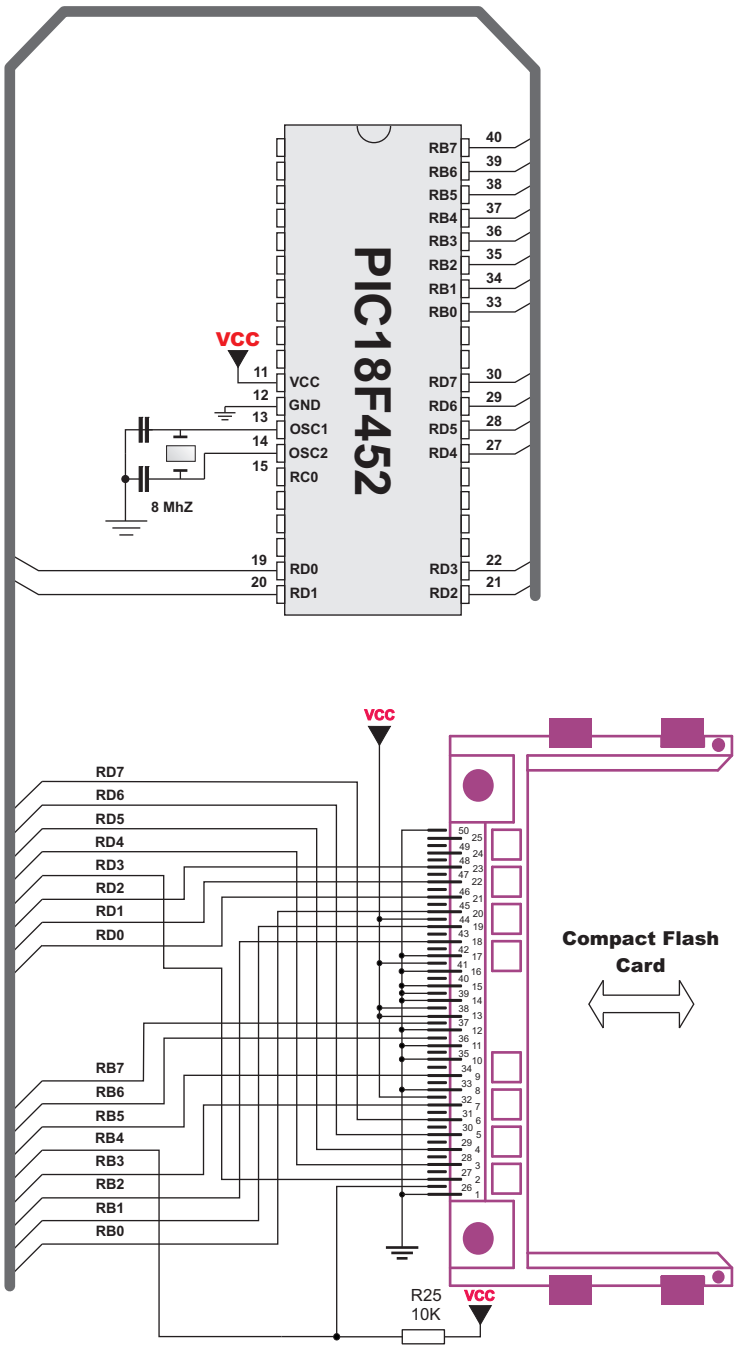
Yapısı	`unsigned long Cf_fat_Get_File_Size();`
Dönüş	Byte olarak dosyanın boyutunu verir.
Tanımı	Bu fonksiyon dosyanın boyutunu byte cinsinden döndürür.
Gereklilikler	Portlar CF ile FAT operasyonları için başlangıç durumuna getirilmelidirler. `Cf_Fat_Init`'e bakınız. Dosya tahsis edilmelidir. `Cf_Fat_Assign`'a bakınız.
Örnek	`Cf_Fat_Get_File_Size;`

Kütüphane Örneği

Aşağıdaki örnek 590 numaralı bölmeye (sector) 512 byte yazıyor ve ardından veriyi okuyup görsel kontrol için PORTC'ye gönderiyor.

```
unsigned i;

void main() {
  TRISC = 0;                    // PORTC'yi cikis yap
  Cf_Init(&PORTB, &PORTD);      // Portlari baslangic durumuna getir

  do nop;
  while (!Cf_Detect());         // CF kart takilana kadar bekle

  Delay_ms(500);
  Cf_Write_Init(590, 1);        // Yazma baslangici 590'uncu bolme olacak
  // 590 no'lu bolmeye 512 bayt yaz
  for (i = 0; i < 512; i++) Cf_Write_Byte(i + 11);

  PORTC = 0xFF;
  Delay_ms(1000);
  Cf_Read_Init(590, 1);         // Okuma baslangici 590'uncu bolme olacak

  // 590 no'lu bolmeden 512 byte oku
  for (i = 0; i < 512; i++) {
    PORTC = Cf_Read_Byte();     // Bayti oku ve C port'unda  goster
    Delay_ms(1000);
  }
}
```

Donanım Bağlantısı



Compact Flash FAT Kütüphanesi v2.xx

Bu, versiyon (v2.1) dan önceki Compact Flash FAT kütüphanesidir. Kullanıcıların eski CF kütüphanesi ile geliştirdikleri projeler sebebiyle bu kütüphane verilmiştir.

NOT: Compact Flash FAT kütüphanesinin bu versiyonu tavsiye edilmemektedir. Kütüphanenin bu versiyonunda daha fazla gelişmeler olmayacaktır. Projeleriniz için Compact Flash kütüphanesinin yeni versiyonunu kullanın.

Önemli! Dosya erişim yordamları dosyayı yazabilir. Dosya isimleri tam 8 karakter uzunluğunda ve BÜYÜK HARF'li olmalıdır. Kullanıcı her dosya için farklı isimler kullandığından emin olmalıdır. Çünkü CF yordamları uygun birleşimleri kontrol etmeyeceklerdir.

Önemli! Yazma işleminden önce, kök(boot) veya FAT bölmesinin üzerine yazmadığınıza emin olun. Aksi takdirde kartınız dijital kamerada veya PC'de okunmaz ve erişilmez olur. Böylesi durumlarda sürücü bellek organizasyonu yazılımları; örneğin Winhex bu konuda size yardımcı olabilir.

Kütüphane Yordamları

```
Cf_Find_File
Cf_File_Write_Init
Cf_File_Write_Byte
Cf_Read_Sector
Cf_Write_Sector
Cf_Set_File_Date
Cf_File_Write_Complete
```

Cf_Find_File

Yapısı	`void Cf_Find_File(unsigned short find_first, char *file_name);`
Tanımı	Yordam, CF kart üzerinde dosyaları arar. `find_first` parametresi sıfır veya sıfır olmayan bir değer olabilir. Eğer sıfır olmayan bir değer ise, yordam kart üzerindeki fiziksel yazım ile düzenlenmiş ilk dosyaya bakar. Değilse bir sonraki dosyayı bulmak için mevcut konumdan ileriye, yine fiziksel düzen içerisinde hareket eder. Eğer dosya bulunursa, yordam onun ismini ve uzantısını `file_name` içine yazar. Eğer hiç dosya bulunmazsa dizi 0'lar ile doldurulur.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.

Örnek	`Cf_Find_File(1, file);`
	`if (file[0]) {  // Eger ilk dosya bulunursa, onu isle`
	`}`

Cf_File_Write_Init

Yapısı	`void Cf_File_Write_Init(void);`
Tanımı	CF kartı yazma işlemi için başlangıç durumuna getirir (sadece FAT16 için).
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`Cf_File_Write_Init();`

Cf_File_Write_Byte

Yapısı	`void Cf_File_Write_Byte(char data);`
Tanımı	Dosyaya bir byte (`data`) veri ekler. ASCII değeri parametre gibi verebilirsiniz; örneğin sıfır için:48
Gereklilikler	CF kartı yazma işlemi için başlangıç durumuna getirilmiş olmalıdır. `Cf_File_Write_Init`'e bakınız.
Örnek	`// Dosyaya 50000 tane sifir (bayt olarak) yaz:` `for (i = 0; i < 50000; i++) Cf_File_Write_Byte(48);`

Cf_Read_Sector

Yapısı	`void Cf_Read_Sector(int sector_number, unsigned short *buffer);`
Tanımı	`sector_number` ile verilen bölmeyi tampon belleğe (buffer) okur.
Gereklilikler	Portların başlangıç ayarları yapılmış olmalıdır. `Cf_Init`'e bakınız.
Örnek	`Cf_Read_Sector(22, data);`

Cf_Write_Sector

Yapısı	`void Cf_Write_Sector(int sector_number, unsigned short *buffer);`
Tanımı	`buffer`' daki değeri, CF'in `sector_number` ile başlayan bölmesine yazar.
Gereklilikler	CF kart yazma işlemi için başlangıç durumuna getirilmiş olmalıdır. `Cf_File_Write_Init`'e bakınız.
Örnek	`Cf_Write_Sector(22, data);`

Cf_Set_File_Date

Yapısı	`void Cf_Set_File_Date(int year, char month,day,hours,min,sec);`
Tanımı	Sistem zaman belirleyicisini dosyaya yazar. Bu yordamı, dosyayı sonlandırmadan önce muhakkak kullanınız; aksi halde, dosya rastgele bir zaman belirleyicisi ile belirtilir.
Gereklilikler	CF kart yazma işlemi için başlangıç durumuna getirilmiş olmalıdır. `Cf_File_Write_Init`'e bakınız.
Örnek	`// 1 Nisan 2005, 18:07:00` `Cf_Set_File_Date(2005,4,1,18,7,0);`

Cf_File_Write_Complete

Yapısı	`void Cf_File_Write_Complete(char filename[8], char *extension);`
Tanımı	Dosyaya yazımı sonlandırır. Tüm veri dosya üzerine yazıldıktan sonra, bu fonksiyon dosyayı kapama ve okunabilir hale getirmek için kullanılır. `filename` parametresi 8 karakter uzunluğunda ve büyük harflerle verilmelidir.
Gereklilikler	CF kart yazma işlemi için başlangıç durumuna getirilmiş olmalıdır. `Cf_File_Write_Init`'e bakınız.
Örnek	`Cf_File_Write_Complete("MY_FILE1","txt");`

EEPROM Kütüphanesi

Bazı PIC MCU'larda EEPROM veri hafızası mevcuttur. mikroC, EEPROM'larla daha rahat çalışabilmek için kütüphane içermektedir.

Kütüphane Yordamları

```
Eeprom_Read
Eeprom_Write
```

Eeprom_Read

Yapısı	`unsigned short Eeprom_Read(unsigned char Address);`
Dönüş	Belirtilen adresteki bir baytlık veriyi döndürür.
Tanımı	Veriyi belirtilen adresten okur.
Gereklilikler	EEPROM birimi gereklidir. `Eeprom_Write` ve `Eeprom_Read` yordamlarının en verimli şekilde kullanımı için aralarında minimum 20 ms gecikme olmasına dikkat ediniz. Aksi halde her ne kadar PIC gerçek değeri yazsa da, `Eeprom_Read` belirsiz değerler döndürebilir.

Örnek	`char take;`
	`...`
	`take = Eeprom_Read(0x3F);`

Eeprom_Write

Yapısı	`void Eeprom_Write(unsigned char Address, unsigned char Data);`
Tanımı	Veriyi belirtilen adrese yazar. `Eeprom_ Write` yordamı çalıştığı zaman tüm kesmeler etkisiz hale gelecektir. (INTCON yazmacının GIE biti sıfırlanacaktır). Yordam bu biti çıkışta tekrar 1 yapar.
Gereklilikler	EEPROM birimi gereklidir. `Eeprom_Write ve Eeprom_Read` yordamlarının en verimli şekilde kullanımı için aralarında minimum 20 ms gecikme olmasına dikkat ediniz. Aksi halde her ne kadar PIC gerçek değeri yazsa da, `Eeprom_Read` belirsiz değerler döndürebilir.
Örnek	`Eeprom_Write(0x32);`

Kütüphane Örneği

```
unsigned short i = 0, j = 0;

void main() {
  PORTB = 0;
  TRISB = 0;

  // Once eeproma 20 baytlik veri yazalim
  j = 4;
  for (i = 0; i < 20u; i++)
    Eeprom_Write(i, j++);

  // Simdi de eeprom'dan 20 baytlik veri okuyup, PORTB'de gosterelim
  for (i = 0; i < 20u; i++) {
    PORTB = Eeprom_Read(i);
    Delay_ms(500);
  }
} //~!
```

Ethernet Kütüphanesi

Bu kütüphane Ethernet işlemlerinde kullanılan donanımın (RTL8019AS) basitleştirilmesi için hazırlanmıştır. Ancak yine de, kullanıcının; ethernet ve ethernet tabanlı protokoller (ARP, IP, TCP/IP, UDP/IP, ICMP/IP) hakkında ön bilgiye sahip olması beklenmektedir. Ethernet çok hızlı ve kullanışlı bir protokoldür. Fakat basit bir protokol değildir. Ancak Ethernet protokolünü öğrendiğinizde PIC projelerinizi CAN veya RS232/485 arabirimi ile yaptığınızdan çok daha geniş bir kitleye ulaştırabileceksiniz.

Kütüphane Yordamları

```
Eth_Init
Eth_Set_Ip_Address
Eth_Inport
Eth_Scan_For_Event
Eth_Get_Ip_Hdr_Len
Eth_Load_Ip_Packet
Eth_Get_Hdr_Chksum
Eth_Get_Source_Ip_Address
Eth_Get_Dest_Ip_Address
Eth_Arp_Response
Eth_Get_Icmp_Info
Eth_Ping_Response
Eth_Get_Udp_Source_Port
Eth_Get_Udp_Dest_Port
Eth_Get_Udp_Port
Eth_Set_Udp_Port
Eth_Send_Udp
Eth_Load_Tcp_Header
Eth_Get_Tcp_Hdr_Offset
Eth_Get_Tcp_Flags
Eth_Set_Tcp_Data
Eth_Tcp_Response
```

Eth_Init

Yapısı	`void Eth_Init(char *addrP, char *dataP, char *ctrlP, char pinReset, char pinIOW, char pinIOR);`
Tanımı	Ethernet kart ve kütüphanesinin başlatılmasını gerçekleştirir: - Kontrol ve veri portlarını ayarlar, - Ethernet kartını (genelde Network Interface Card veya kısaca NIC olarak bilinir) başlatır, - NIC donanımının (MAC) adresine ulaşır ve yerel olarak saklar, - NIC'i dinleme (LISTEN) moduna getirir, `addrP` parametresi adres portunun adresleri için bir işaretleyicidir (pointer). `dataP` parametresi ise veri portu için bir işaretleyicidir. `ctrlP` parametresi kontrol portudur. `pinReset` parametresi ethernet kartı üzerindeki yonganın reset/enable pinidir. `pinIOW` parametresi I/O yazma talebi kontrol pinidir. `pinIOR` parametresi I/O okuma talebi kontrol pinidir.
Gereklilikler	Tüm kütüphane için yukarıda belirtildiği gibidir.
Örnek	`Eth_Init(&PORTB, &PORTD, &PORTE, 2, 1, 0);`

Eth_Set_Ip_Address

Yapısı	`void Eth_Set_Ip_Address(char ip1, char ip2, char ip3, char ip4);`
Tanımı	Bağlanılan ve başlatılan ethernet ağ kartının IP adresini ayarlar. Fonksiyonun bağımsız değişkeni; IPv4 formatındaki IP sayılardır.(örneğin 127.0.0.1 gibi)
Gereklilikler	Bu yordam NIC başlangıç ayarı yapıldıktan hemen sonra çağırılmalıdır (`Eth_init`'e bakınız). Herhangi bir anda, kodun herhangi bir yerinde, IP adresinizi değiştirebilirsiniz.
Örnek	`// IP adresini 192.168.20.25'e ayarla` `Eth_Set_Ip_Address(192u, 168u, 20u, 25u);`

Eth_Inport

Yapısı	`unsigned short Eth_Inport(unsigned short address);`
Dönüş	Belirtilen adresten bir bayt döndürür.
Tanımı	Ethernet kart entegresinin belirtilen adresinden bir bayt döndürür.
Gereklilikler	Kart (NIC) uygun şekilde başlatılmalıdır. `Eth_Init`'e bakınız.
Örnek	`udp_length |= Eth_Inport(NIC_DATA);`

Eth_Scan_For_Event

Yapısı	`unsigned Eth_Scan_For_Event(unsigned short *next_ptr);`
Dönüş	Alınan ethernet paket'inin tipidir. İki ayrı tip şunlardır: `ARP` (MAC-IP address data request) ve `IP` (Internet Protocol).
Tanımı	Göndericinin MAC adresi ve paketin tipi alınır. Fonksiyonun bağımsız değişkeni RTL8019 tamponunun içerisindeki bir sonraki veri paketinin işaretçisidir ve son kullanıcı için önemi yoktur.
Gereklilikler	NIC Kartı uygun şekilde başlatılmalıdır. `Eth_Init`'e bakınız. Ayrıca fonksiyon uygun bir sıralama içerisinde çağırılmalıdır. Örneğin: kartın ve IP address/UDP port'unun başlatılmasından hemen sonra.

Örnek	`Eth_Init(&PORTB, &PORTD, &PORTE, 2, 1, 0);`
	`Eth_Set_Ip_Address(192u, 168u, 20u, 25u);`
	`Eth_Set_Udp_Port(10001);`
	`do { // Ethernet orneginin ana blogu`
	`  event_type = Eth_Scan_For_Event(&next_ptr);`
	`  if (event_type) {`
	`  switch (event_type) { case ARP: Arp_Event(); break;`
	`  case IP : Ip_Event();}`
	`  Eth_Outport(CR, 0x22);`
	`  Eth_Outport(BNDRY, next_ptr);`
	`  }`
	`} while (1);`

Eth_Get_Ip_Hdr_Len

Yapısı	`unsigned short Eth_Get_Ip_Hdr_Len(void);`
Dönüş	Alınan IP paketinin başlık uzunluğunu (length) döndürür.
Tanımı	Fonksiyon alınan IP paketinin başlık (header) uzunluğunu döndürür. IP protokol'ü üzerindeki (TCP, UDP, ICMP) diğer veriler analiz edilmeden önce, alınan IP paketindeki alt protokol verisi doğru olarak yüklenmelidir.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init`'e bakınız. Fonksiyon uygun bir sıralama içerisinde çağırılmalıdır. Örneğin: Alınan paketin IP paketi olduğu belirlendikten hemen sonra.
Örnek	`// IP basligini al` `opt_len = Eth_Get_Ip_Hdr_Len() - 20;`

Eth_Load_Ip_Packet

Yapısı	`void Eth_Load_Ip_Packet(void);`
Tanımı	Çeşitli IP paket verilerini PIC'in Ethernet değişkenlerine yükler.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init`'e bakınız. Aynı zamanda uygun fonksiyon çağırma sıralamasına dikkat edilmelidir. (Ethernet örneğindeki `IP_Event` fonksiyonuna bakınız.)
Örnek	`Eth_Load_Ip_Packet();`

Eth_Get_Hdr_Chksum

Yapısı	`void Eth_Get_Hdr_Chksum(void);`
Tanımı	Alınan IP paketinin başlığının sağlama toplamını (checksum) yükler ve döner.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init'` e bakınız. Aynı zamanda uygun fonksiyon çağırma sıralamasına dikkat edilmelidir. (Ethernet örneğindeki `IP_Event` fonksiyonuna bakınız.)
Örnek	`Eth_Get_Hdr_Chksum();`

Eth_Get_Source_Ip_Address

Yapısı	`void Eth_Get_Source_Ip_Address(void);`
Tanımı	Alınan IP paketinin göndericisinin IP adresini yükler ve döner.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init`'e bakınız. Aynı zamanda uygun fonksiyon çağırma sıralamasına dikkat edilmelidir. (Ethernet örneğindeki `IP_Event` fonksiyonuna bakınız.)
Örnek	`Eth_Get_Source_Ip_Address();`

Eth_Get_Dest_Ip_Address

Yapısı	`void Eth_Get_Dest_Ip_Address(void);`
Tanımı	Alınan IP paketinin IP adresini gösterilen paket için yükler.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init`'e bakınız. Aynı zamanda uygun fonksiyon çağırma sıralamasına dikkat edilmelidir. (Ethernet örneğindeki `IP_Event` fonksiyonuna bakınız.)
Örnek	`Eth_Get_Dest_Ip_Address();`

Eth_Arp_Response

Yapısı	`void Eth_Arp_Response(void);`
Tanımı	Bir otomatik ARP yanıtıdır. Kullanıcı NIC içerisinde ARP olayını algılayınca bu fonksiyonu çağırmalıdır.
Gereklilikler	Tüm kütüphane için tanımlandığı gibi.
Örnek	`Eth_Arp_Response();`

Eth_Get_Icmp_Info

Yapısı	`void Eth_Get_Icmp_Info(void);`
Tanımı	Alınan ICMP paketinin başlığından ICMP protokol bilgisini yükler ve bu bilgiyi PIC'in Ethernet değişkenlerine depolar.
Gereklilikler	Kart (NIC) doğru olarak başlatılmalıdır. `Eth_Init`'e bakınız. Aynı zamanda uygun fonksiyon çağırma sıralamasına dikkat edilmeli ve `Eth_Ping_Response`'dan önce çağırılmalıdır.
Örnek	`Eth_Get_Icmp_Info();`

Eth_Ping_Response

Yapısı	`void Eth_Ping_Response(void);`
Tanımı	Bir otomatik ICMP (ping) yanıtıdır. Kullanıcı bu fonksiyonu bir ICMP/IP olayına cevap vereceği zaman kullanmalıdır.
Gereklilikler	Tüm kütüphane için tanımlandığı gibi.
Örnek	`Eth_Ping_Response();`

Eth_Get_Udp_Source_Port

Yapısı	`unsigned Eth_Get_Udp_Source_Port(void);`
Dönüş	Alınan UDP paketinin kaynak portunu (soket) döndürür.
Tanımı	Fonksiyon alınan UDP paketinin kaynak portunu döndürür. Geçerli bir IP paketinin alındığının algılanmasından ve tipinin UDP olduğu kararlaştırıldıktan sonra, UDP paketi başlığı yorumlanmalıdır. UDP kaynak portu UDP başlık alanının ilk verisidir.
Gereklilikler	Bu fonksiyon uygun bir sıralama içerisinde çağırılmalıdır. Örneğin IP paket başlığının yorumlanmasından hemen sonra (UDP paket başlığı alımının hemen başında).
Örnek	`udp_source_port = Eth_Get_Udp_Source_Port();`

Eth_Get_Udp_Dest_Port

Yapısı	`unsigned Eth_Get_Udp_Dest_Port(void);`
Dönüş	Alınan UDP paketinin hedef portunu döndürür.
Tanımı	Fonksiyon alınan UDP paketinin hedef portunu döndürür. UDP paket başlığının içeriğindeki ikinci bilgi paketin gönderileceği hedef portudur.
Gereklilikler	Bu fonksiyon uygun bir sıralama içerisinde çağırılmalıdır. Örneğin `ETH_Get_Udp_Source_Port` fonksiyonu çağırıldıktan hemen sonra.
Örnek	`udp_dest_port = Eth_Get_Udp_Dest_Port();`

Eth_Get_Udp_Port

Yapısı	`unsigned short Eth_Get_Udp_Port(void);`
Dönüş	PIC Ethernet kartı için ayarlanmış olan UDP port numaralarını döndürür.
Tanımı	Fonksiyon PIC Ethernet kartı için kurulmuş olan UDP port numarasını döndürür. Yeni oturumun başında UDP portu ayarlandıktan sonra (`Eth_Set_Udp_Port`), bu numara daha sonra alınan UDP paketinin bizim kullandığımız portu hedefleyip hedeflemediğini test etmek için kullanılır.
Gereklilikler	Ağ kartı düzgün bir şekilde başlatılmalıdır (`Eth_Init`'e bakınız) ve UDP portu tam olarak ayarlanmalıdır (`Eth_Set_Udp_Port`'a bakınız). Bu kütüphane aynı anda sadece bir UDP portunun çalışmasını desteklemektedir.

Örnek	`if (udp_dest_port == Eth_Get_Udp_Port()) {`
	`  ... // Bu mesaj bize, haydi karsilik verelim`
	`}`

Eth_Set_Udp_Port

Yapısı	`void Eth_Set_Udp_Port(unsigned udp_port);`
Tanımı	Kullanıcı isteklerine cevap verecek geçerli UDP portunu ayarlar. Kullanıcı, alınan UDP paketi üzerinden, bu paketin hangi porta yollandığına ve kabul veya red edileceğine karar verebilir.
Gereklilikler	Tüm kütüphane için belirtidiği gibi
Örnek	`Eth_Set_Udp_Port(10001);`

Eth_Send_Udp

Yapısı	`void Eth_Send_Udp(char *msg);`
Tanımı	16 byte uzunluğuna kadar hazırlanmış UDP mesajını (`msg`) yollar. ICMP ve TCP protokollerinin aksine, UDP paketleri genellikle istemci tarafa bir cevap olarak gönderilmez. UDP; mesajın ulaşıp ulaşmaması ile ilgili bir garanti vermez ve gönderici ağa gönderilen bir UDP mesajıyla ilgili bir durum bilgisi tutmaz. Bu, UDP paketlerinin kullanıcı isteklerine cevap olmayıp sadece gönderilmesinin de cevabıdır.
Gereklilikler	Tüm kütüphane için belirtildiği gibi. Ayrıca gönderilen mesaj hiçlik-sonlandırıcılı (null terminated) bir dizi gibi biçimlendirilmelidir. Mesaj uzunluğu, (arkadaki "0" dahil) 16 karakteri geçmemelidir.
Örnek	`Eth_Send_Udp(udp_tx_message);`

Eth_Load_Tcp_Header

Yapısı	`void Eth_Load_Tcp_Header(void);`
Tanımı	Değişik TCP başlık verilerini PIC'in Ethernet değişkenlerine yükler.
Gereklilikler	Bu fonksiyon düzgün bir sıralama ile çağırılmalıdır. Örneğin TCP mesajının kaynak ve hedef portlarının alımından hemen sonra.

Örnek	`// Kaynak portu ogren`
	`tcp_source_port  = Eth_Inport(NIC_DATA) << 8;`
	`tcp_source_port |= Eth_Inport(NIC_DATA);`
	`// Hedef portu ogren`
	`tcp_dest_port  = Eth_Inport(NIC_DATA) << 8;`
	`tcp_dest_port |= Eth_Inport(NIC_DATA);`
	
	`// sadece port 80'e yanit ver (HTML talebi)`
	`if (tcp_dest_port == 80u) {`
	`  Eth_Load_Tcp_Header(); // TCP baslik verisini al (cogunu)`
	`  //...`
	`}`

Eth_Get_Tcp_Hdr_Offset

Yapısı	`unsigned short Eth_Get_Tcp_Hdr_Offset(void);`
Dönüş	TCP paket başlığının boyutunu (veya ofsetini) byte olarak döndürür.
Tanımı	Fonksiyon TCP paket başlığının boyutunu (veya ofsetini) byte olarak döndürür. Geçerli bir TCP paketinin alınmasından sonra düzgün bir şekilde yanıtlama yapabilmek için paket başlığının analiz edilmesi gerekir (diğer tarafın isteklerine yanıt, birkaç paketin aynı mesaja birleştirilmesi vs.). Başlıktan bilginin çıkarılabilmesi için başlık boyutunun bilinmesi önemlidir.
Gereklilikler	Bu fonksiyon `Eth_Load_Tcp_Header`'dan sonra çağırılmalıdır, zira bu yordam bu fonksiyon için özel bazı değişkenlerin ilk değerlerini ayarlar.
Örnek	`// Ofset'i hesapla (TCP baslik boyutu)` `tcp_options = Eth_Get_Tcp_Hdr_Offset() - 20;`

Eth_Get_Tcp_Flags

Yapısı	`unsigned short Eth_Get_Tcp_Flags(void);`
Dönüş	Alınan TCP paketinin başlığından bayrak (flag) verisini döndürür.
Tanımı	Fonksiyon alınan TCP paketinin başlığından bayrak (flag) verisini döndürür. TCP bayrakları değişik bilgileri gösterirler, mesela; SYN (syncronize request), ACK (acknowledge receipt) gibi. Bu bayraklar yardımıyla uygun bir HTTP iletişimi kurulabilir.
Gereklilikler	Bu fonksiyon `Eth_Load_Tcp_Header`'dan sonra çağırılmalıdır, zira bu yordam bu fonksiyon için özel bazı değişkenlerin ilk değerlerini ayarlar.
Örnek	`flags = Eth_Get_Tcp_Flags();`

Eth_Set_Tcp_Data

Yapısı	`void Eth_Set_Tcp_Data(const unsigned short *data);`
Tanımı	HTTP isteği durumunda gönderilecek veriyi hazırlar. Bu kütüphane sadece HTTP isteklerini kontrol edebilir, bu nedenle diğer TCP tabanlı protokollerin gönderiminde, örneğin FTP'de hataya yol açar. TCP/IP protokolü 8-bitlik mikrodenetleyiciler göz önüne alınarak tasarlanmadığından, HTTP isteklerinizde ölçülü davranınız.
Gereklilikler	Tüm kütüphane için belirtildiği gibi.

Örnek	`// Karakter dizisinde basit bir HTML sayfasi hazirlayalim:`
	`const char httpPage1[] =`
	`  "HTTP/1.0 200 OK\nContent-type: text/html\n"`
	`  "<html>\n" "<body>\n"`
	`  "<h1>Merhaba dunya!</h1>\n"`
	`  "</body>\n" "</html>";`
	`//...`
	`Eth_Set_Tcp_Data(httpPage1);`
	`//...`

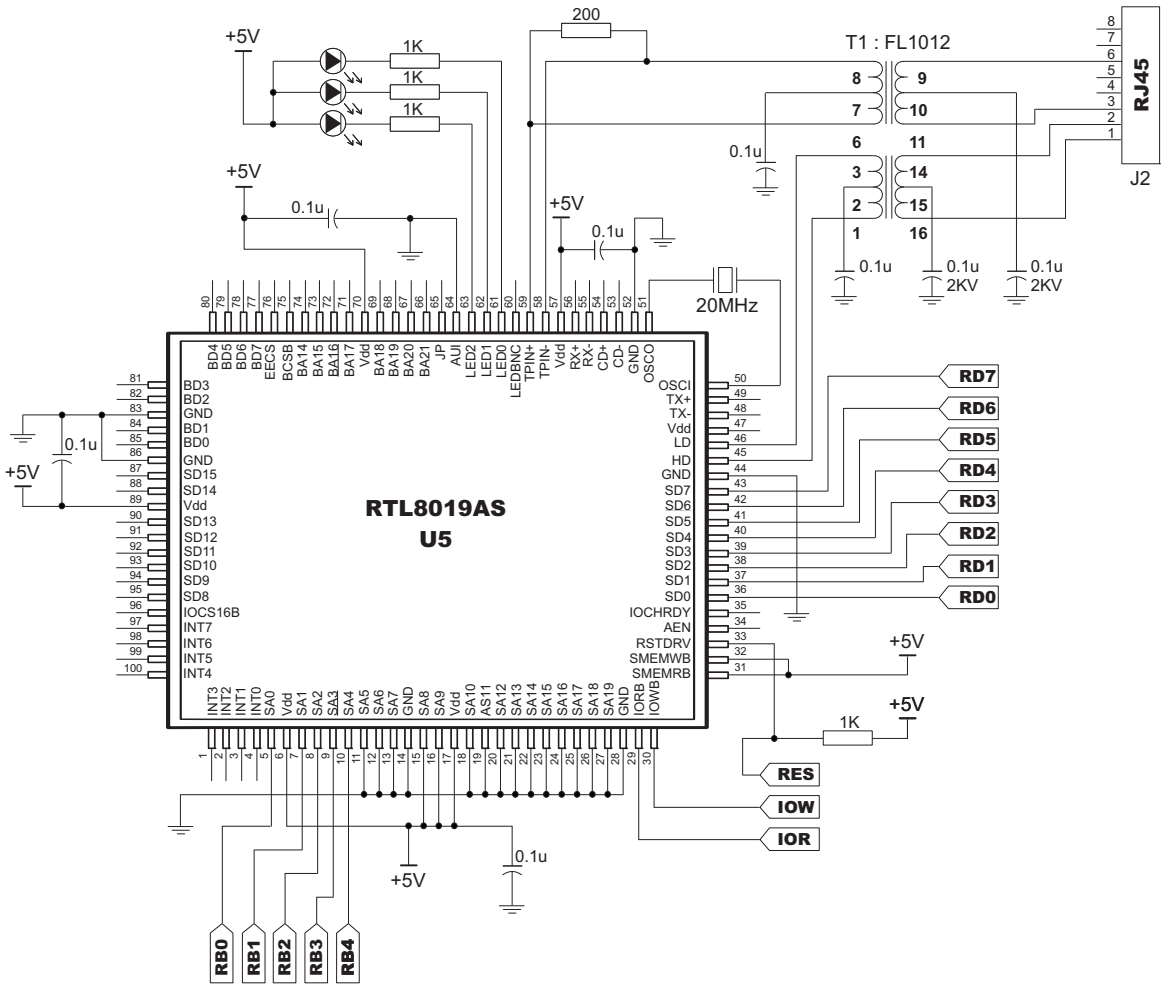
Eth_Tcp_Response

Yapısı	`void Eth_Tcp_Response(void);`
Tanımı	TCP/IP olayına kullanıcının yanıtını gerçekleştirir. Kullanıcı, alınan isteğe bağlı olarak (HTTP, HTTPD, FTP gibi) yollanacak veriyi belirler. Bu işlem `Eth_Set_Tcp_Data` fonksiyonu sayesinde yerine getirilir.
Gereklilikler	Donanım gereklilikleri tüm kütüphane için belirtildiği gibidir. Bu yordamın kullanımından önce kullanıcı, yollanacak veriyi TCP üzerinden hazırlamalıdır. (`Eth_Set_Tcp_Data`'ya bakınız).
Örnek	`Eth_Tcp_Response();`

Kütüphane Örneği

CD'nizdeki "Examples" dizinindeki Ethernet örneğini inceleyiniz.

Donanım Bağlantısı



SPI Ethernet Kütüphanesi

ENC28J60 tümleşik entegresi; endüstri standardı olmuş SPI arabirimi ihtiva eden bir ethernet denetleyicisidir. Üzerinde SPI ihtiva eden herhangi bir denetleyici ile Ethernet arabirimi kurmak üzere tasarlanmıştır.

ENC28J60 IEEE 802.3 standardının tüm özelliklerini sağlar. Entegre üzerinde, gelen veri paketlerini kısıtlamak için, paket filtreleme devreleri vardır. Aynı zamanda hızlı veri akışı ve donanım destekli IP sağlama toplaması hesaplamaları için bir iç DMA birimi bulundurur. Ana sistem denetleyicisi ile haberleşme, iki kesme ve SPI ile 10 Mb/s'a ulaşan veri aktarım hızı ile gerçekleştirilir. Entegrenin iki pini LED için hat ve ağ aktivite göstergesi olarak kullanılmak üzere ayrılmıştır.

Bu kütüphane ENC28J60 donanımınının kullanımını daha basit hale getirmek amacıyla oluşturulmuştur. Kütüphane 4 Kb'den daha fazla ROM'a sahip ve tümleşik SPI özelliği içeren PIC'ler ile çalışmaktadır. 8 ile 10 MHz SPI saat hızı elde edebilmek için 38 ile 40 MHz arasında bir saat sinyali gerekmektedir. Aksi halde SPI donanımındaki ENC silikon hatasından dolayı, PIC saat sinyali ENC saat çıkışından alınmalıdır. Eğer daha düşük PIC saat hızı kullanırsanız, kartta kitlenmeler oluşabilir veya bazı isteklere cevap verememe hataları oluşabilir. Bu kütüphane 10Mhz saat ile çalışan PIC16F877A ve 40Mhz saat ile çalışan PIC18F452 ile test edilmiştir.

Not: Tecrübeli programcılar, Uses\P16 ve Uses\P18 klasörlerinin altında bulunan ve SPI Ethernet Kütüphane fonksiyonlarını detaylı anlatan bölümden yaralanabilirler (enc28j60_libprivate.h).

Kütüphane Yordamları

Not:SPI ethernet'i başlatmadan önce `SPI_Init` çağırılmalıdır.

```
SPI_Ethernet_Init
SPI_Ethernet_doPacket
SPI_Ethernet_putByte
SPI_Ethernet_getByte
SPI_Ethernet_UserTCP
SPI_Ethernet_UserUDP
```

SPI_Ethernet_Init

Yapısı	`void SPI_Ethernet_Init(unsigned char *resetPort, unsigned char resetBit, unsigned char *CSportPtr, unsigned char CSbit, unsigned char *mac, unsigned char *ip, unsigned char fullDuplex);`
Dönüş	Yok.
Tanımı	SPI ve ENC denetleyicisinin başlangıç ayarlarını yapar. Bu yordam bellek azlığında bağlayıcıya yardımcı olmak üzere iki kısımda işlenir. **`resetPort`** – port'un reset pinini gösteren işaretleyici **`resetBit`** – `resetPort`'undaki reset bitinin numarası **`CSport`** – port'un CS pinini gösteren işaretleyici **`CsBit`** – CSport'undaki CS bitinin numarası **`mac`** – MAC adresini içeren 6 karakterlik dizi işaretleyicisi **`ip`** – ip adresini içeren 4 karakterlik dizi işaretleyicisi **fullDuplex** – ya yarı çift yönlü iletişim için `SPI_Ethernet_HALFDUPLEX` veya tam çift yönlü iletişim için `SPI_Ethernet_FULLDUPLEX`
Gereklilikler	SPI ethernet'i başlatmadan önce `SPI_Init` çağırılmalıdır.
Örnek	`SPI_Ethernet_Init(&PORTC, 0, &PORTC, 1, myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);`

SPI_Ethernet_doPacket

Yapısı	`void SPI_Ethernet_doPacket();`
Dönüş	Yok.
Tanımı	Gelen bir paket varsa onu işler. Bu yordam alenidir.
Gereklilikler	Bu yordamdan önce `SPI_Ethernet_Init` çağırılmış olmalıdır. Bu fonksiyon kullanıcı tarafından mümkün olduğu kadar sık çağrılmalıdır.
Örnek	`SPI_Ethernet_doPacket();`

SPI_Ethernet_putByte

Yapısı	`void SPI_Ethernet_putByte(unsigned char v);`
Dönüş	Yok.
Tanımı	v = saklanacak değer Bir baytı geçerli EWRPT ENC lokasyonuna (yerleşimine) saklar. Bu fonksiyon alenidir.
Gereklilikler	Bu fonksiyondan önce `SPI_Ethernet_Init` çağırılmış olmalıdır.
Örnek	`SPI_Ethernet_putByte(0xa0);`

SPI_Ethernet_getByte

Yapısı	`unsigned char SPI_Ethernet_getByte();`
Dönüş	İlgili adresteki baytın değerini döndürür.
Tanımı	ERDPT ENC lokasyonundaki byte'ı al. Bu yordam alenidir.
Gereklilikler	Bu fonksiyondan önce `SPI_Ethernet_Init` çağırılmış olmalıdır.
Örnek	`b = SPI_Ethernet_getByte();`

SPI_Ethernet_UserTCP

Yapısı	`unsigned int SPI_Ethernet_UserTCP(unsigned char *remoteHost, unsigned int remotePort, unsigned int localPort, unsigned int reqLength);`
Dönüş	HTTP cevabının uzunluğunu bayt olarak verir. Hiç bir şey gönderilmemişse 0 verir.
Tanımı	Bu fonksiyon kütüphane tarafından içsel olarak çağırılır. Kullanıcı HTTP isteklerine `SPI_Ethernet_getByte` yordamını ard arda çağırarak erişebilir. Kullanıcı gönderme tamponuna veriyi `SPI_Ethernet_putByte` yordamını ard arda çağırarak yerleştirebilir. Bu fonksiyon gönderilecek HTTP cevaplarındaki baytların uzunluğunu döndürmelidir. Eğer gönderilecek bir şey yoksa 0 döndürmelidir. Eğer HTTP isteklerine cevap vermeyecekseniz bu fonksiyonu sadece return(0) deyimini içerecek şekilde tanımlayınız.
Gereklilikler	Bu fonksiyondan önce `SPI_Ethernet_Init` çağırılmalıdır.
Örnek	

SPI_Ethernet_UserUDP

Yapısı	`unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost, unsigned int remotePort, unsigned int destPort, unsigned int reqLength);`
Dönüş	UDP cevabının uzunluğunu bayt olarak verir. Hiç bir şey gönderilmemişse 0 verir.
Tanımı	Bu fonksiyon kütüphane tarafından içsel olarak çağırılır. Kullanıcı UDP isteklerine `SPI_Ethernet_getByte()` yordamını ard arda çağırarak erişebilir. Kullanıcı gönderme tamponuna veriyi, `SPI_Ethernet_putByte()` yordamını ard arda çağırarak yerleştirebilir. Bu fonksiyon gönderilecek UDP cevaplarındaki baytların uzunluğunu döndürmelidir. Eğer gönderilecek bir şey yoksa 0 döndürmelidir. Eğer UDP isteklerine cevap vermeyecekseniz bu fonksiyonu sadece return(0) deyimini içerecek şekilde tanımlayınız.
Gereklilikler	Bu fonksiyondan önce `SPI_Ethernet_Init` çağırılmalıdır.
Örnek	

Kütüphane Örneği

Aşağıdaki programda SPI Ethernet kütüphanesinin kullanımına basit bir örnek verilmektedir. PIC'e 192.168.20.60 IP adresi verilmiştir ve bir yerel ağa bağlı ise ping'e cevap verecektir.

```
#define SPI_Ethernet_HALFDUPLEX      0
#define SPI_Ethernet_FULLDUPLEX      1

/************************************************************
 * ROM sabit karakter dizileri
 */
const unsigned char httpHeader[] = "HTTP/1.1 200 OK\nContent-type: " ; // HTTP baslik
const unsigned char httpMimeTypeHTML[] = "text/html\n\n" ;        // HTML MIME tip
const unsigned char httpMimeTypeScript[] = "text/plain\n\n" ;     // TEXT MIME tip
unsigned char httpMethod[] = "GET /";
/*
 * web sayfası 2 ye bolundu :
 * ROM yetersiz geldiginde, ufak parcalara ayrilmis olan veri
 * baglayici tarafindan daha verimli bicimde ele alinabilir.
 *
 * Bu HTML sayfasi, durumunu almak için karta baglanir
 * ve javascript ile kendisini olusturur. */
const char     *indexPage = "<HTML><HEAD></HEAD><BODY>\
<h1>PIC + ENC28J60 Mini Web Server</h1>\
<a href=/>Reload</a>\
<script src=/s></script>\
<table><tr><td valign=top><table border=1 style=\"font-size:20px ;font-family: termi-
nal ;\">\
<tr><th colspan=2>ADC</th></tr>\
<tr><td>AN2</td><td><script>document.write(AN2)</script></td></tr>\
<tr><td>AN3</td><td><script>document.write(AN3)</script></td></tr>\
</table></td><td><table border=1 style=\"font-size:20px ;font-family: terminal ;\">\
<tr><th colspan=2>PORTB</th></tr>\
<script>\
var str,i;\
str=\"\";\
for(i=0;i<8;i++)\
{ str+=\"<tr><td bgcolor=pink>BUTTON #\"+i+\"</td>\";\
if(PORTB&(1<<i)){ str+=\"<td bgcolor=red>ON\";}\
else { str+=\"<td bgcolor=#cccccc>OFF\";}\
str+=\"</td></tr>\";}\
document.write(str) ;\
```

```
</script>\
" ;

const char    *indexPage2 =  "</table></td><td>\
<table border=1 style=\"font-size:20px ;font-family: terminal ;\">\
<tr><th colspan=3>PORTD</th></tr>\
<script>\
var str,i;\
str=\"\";\
for(i=0;i<8;i++)\
{ str+=\"<tr><td bgcolor=yellow>LED #\"+i+\"</td>\";\
if(PORTD&(1<<i)){ str+=\"<td bgcolor=red>ON\";}\
else { str+=\"<td bgcolor=#cccccc>OFF\";}\
str+=\"</td><td><a href=/t\"+i+\">Toggle</a></td></tr>\";}\
document.write(str) ;\
</script>\
</table></td></tr></table>\
This is HTTP request #<script>document.write(REQ)</script></BODY></HTML>\
" ;
//                                    str+=\"</td><td><a
href=/t\"+i+\">Toggle</a></td></tr>\";}\

/**********************************
 * RAM değişkenleri
 */
unsigned char  myMacAddr[ 6]  = { 0x00, 0x14, 0xA5, 0x76, 0x19, 0x3f};//benim MAC adresim
unsigned char  myIpAddr[ 4]  = { 192, 168, 20, 60} ;    // benim IP adresim
unsigned char  getRequest[ 15] ;                        // HTTP istek tamponu
unsigned char  dyna[ 31] ;                              // dinamik cevap için tampon
unsigned long  httpCounter = 0 ;                        // HTTP istegi sayaci

/*****************************************
 * fonksiyonlar
 */

/*
 * ENC gonderim tamponuna, s ile isaretlenen karakter dizisi sabitlerini koy.
 */
```

```
unsigned int     putConstString(const char *s)
         {
         unsigned int ctr = 0 ;

         while(*s)
                  {
                  SPI_Ethernet_putByte(*s++) ;
                  ctr++ ;
                  }
         return(ctr) ;
         }

/*
 * ENC gonderim tamponuna, s ile isaretlenen karakter dizisi sabitlerini koy.
 */
unsigned int     putString(char *s)
         {
         unsigned int ctr = 0 ;

         while(*s)
                  {
                  SPI_Ethernet_putByte(*s++) ;
                  ctr++ ;
                  }
         return(ctr) ;
         }

/*
 * Bu fonksiyon kutuphane tarafindan çagirilir
 * Kullanici HTTP istegine ard arda SPI_Ethernet_getByte() cagrilari
 * yaparak erisebilir.
 * Kullanici veriyi iletim tamponunun icine ard arda SPI_Ethernet_putByte()
 * cagrilari yaparak koyabilir.
 * Fonksiyon HTTP yanitinin uzunlugunu bayt cinsinden dondurmelidir.
 * Veya gonderilecek hic veri yok ise 0 dondurmelidir.
 *
 * Eger gelen HTTP istegine cevap gonderme ihtiyaciniz yoksa, bu fonksiyonu
 * sadece return(0) deyimini icerecek sekilde tanimlayiniz.
 *
 */
```

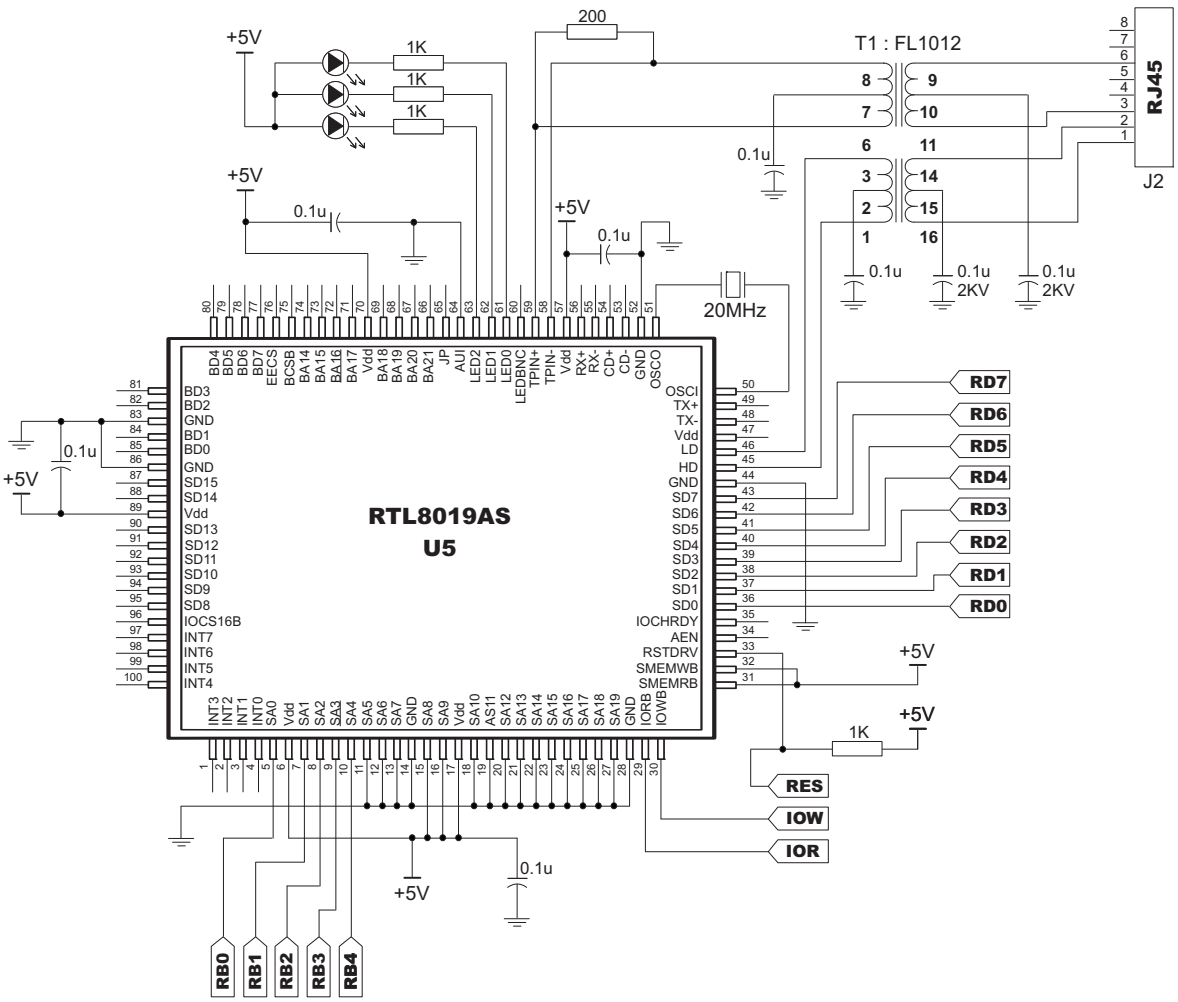
```
unsigned int     SPI_Ethernet_UserTCP(unsigned char *remoteHost, unsigned int
remotePort, unsigned int localPort, unsigned int reqLength)
        {
        unsigned int     len = 0 ;      // Cevabimin uzunlugu
        unsigned int     i ;            // Genel amacli tamsayi

        if(localPort != 80)      // Sadece Port 80'den gelen web istegini dinliyorum.
                {
                return(0) ;
                }

        // Istegin sadece ilk 10 byte'ini al, gerisi burada onemli degil
        for(i = 0 ; i < 10 ; i++)
                {
                getRequest[ i] = SPI_Ethernet_getByte() ;
                }
        getRequest[ i] = 0 ;

        if(memcmp(getRequest, httpMethod, 5)) // Burada sadece GET metodu desteklenir
                {
                return(0) ;
                }

        httpCounter++ ;          // bir tane daha istek yapildi

        if(getRequest[ 5] == 's')
// Eger istegin "yol ismi" s ile baslar ise, dinamik veriyi iletim tamponuna sakla
                {
/* Bu istege cevap olan metin karakter dizisi,
 * internet tarayicisi programlar tarafindan
 * javascript olarak yorumlanabilir. */



                len = putConstString(httpHeader) ;          // HTTP basligi
                len += putConstString(httpMimeTypeScript) ; // MIME metin tipiyle

                // Cevaba AN2 degerini ekle
                intToStr(ADC_Read(2), dyna) ;
                len += putConstString("var AN2=") ;
                len += putString(dyna) ;
                len += putConstString(";");
```

```
                // Cevaba AN3'un degerini ekle
                intToStr(ADC_Read(3), dyna) ;
                len += putConstString("var AN3=") ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // Cevaba PORTB'nin (butonlar) degerini ekle
                len += putConstString("var PORTB=") ;
                intToStr(PORTB, dyna) ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // Cevaba PORTD'nin (Ledler) degerini ekle
                len += putConstString("var PORTD=") ;
                intToStr(PORTD, dyna) ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // Cevaba HTTP istek sayacini ekle
                intToStr(httpCounter, dyna) ;
                len += putConstString("var REQ=") ;
                len += putString(dyna) ;
                len += putConstString(";") ;
                }
        else if(getRequest[ 5]  == 't')
/* Eger istegin "yol ismi" t ile baslar ise,
 * sonrasinda gelen PORTD (led) bit nosunu tersle (toggle) */
                {
                unsigned char   bitMask = 0 ;          // bit maskeleri için

                if(isdigit(getRequest[ 6] ))
// Eger 0 <= bit sayisi <= 9 ise, 8 ve 9 uncu bitler mevcut degil, fakat sorun degil
{
                        bitMask = getRequest[ 6]  - '0' ; // ASCII'yi int'e donustur
                        bitMask = 1 << bitMask ;    // bit maskesi olustur
                        PORTD ^= bitMask ;    // PORTD'yi xor operatörü ile tersle
                        }
                }
```

```
        if(len == 0)                                    // varsayilan olarak yapilacaklar
                  {
                  len =  putConstString(httpHeader) ;      // HTTP basligi
                  len += putConstString(httpMimeTypeHTML) ;// HTML MIME tipi ile
                  len += putConstString(indexPage) ;       // HTML sayfasi ilk bolum
                  len += putConstString(indexPage2) ;      // HTML sayfasi 2.nci bolum
                  }

         return(len) ; // Iletilecek baytlarin sayisini kutuphaneye dondur.
         }
/*
 * Bu fonksiyon kutuphane tarafindan cagirilir.
 * Kullanici UDP istegine ard arda SPI_Ethernet_getByte()'i cagrilari yaparak
 * erisebilir.
 * Kullanici veriyi iletim tamponunun icine ard arda SPI_Ethernet_putByte()
 * yaparak koyabilir.
 * Fonksiyon UDP yanitinin uzunlugunu bayt cinsinden dondurmelidir.
 * Veya gonderilecek hic veri yok ise 0 dondurmelidir.
 *
 * Eger gelen UDP istegine cevap gonderme ihtiyaciniz yoksa, bu fonksiyonu
 * sadece return(0) deyimini icerecek sekilde tanimlayiniz.
 *

 *
 */
unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost, unsigned int remotePort,
unsigned int destPort, unsigned int reqLength)
         {
         unsigned int     len ;                          // Cevabimin uzunlugu
         unsigned char    *ptr ;                         // Dinamik tampona isaretci

         // Cevap okunabilir formatta "uzak sistem" IP adresinden uretilir.
         byteToStr(remoteHost[ 0], dyna) ;               // Ilk IP adres byte'i
         dyna[ 3]  = '.' ;
         byteToStr(remoteHost[ 1], dyna + 4) ;           // Ikinci
         dyna[ 7]  = '.' ;
         byteToStr(remoteHost[ 2], dyna + 8) ;           // Ucuncu
         dyna[ 11]  = '.' ;
         byteToStr(remoteHost[ 3], dyna + 12) ;          // Dorduncu

         dyna[ 15]  = ':' ;                              // Ayrac ekle

         // Ve daha sonra remote host port numarası
         intToStr(remotePort, dyna + 16) ;
         dyna[ 22]  = '[ ' ;
         intToStr(destPort, dyna + 23) ;
         dyna[ 29]  = '] ' ;
         dyna[ 30]  = 0 ;
```

```
// Yanitin toplam uzunlugu, dinamik karakter dizileri + talep metninin uzunlugudur
        len = 30 + reqLength ;
        // Dinamik karakter dizisini iletim tamponuna koyar.
        ptr = dyna ;
        while(*ptr)
                {
                SPI_Ethernet_putByte(*ptr++) ;
                }
/* Daha sonra BUYUK HARFLERE donusturulmus talep karakter dizilerini iletim
tamponuna koyar. */
        while(reqLength--)
                {
                SPI_Ethernet_putByte(toupper(ENC28J60_getByte())) ;
                }

        return(len) ;    // UDP yanitinin boyutuyla kutuphaneye donus
        }
/*
 * Ana giris noktasi
 */
void    main()
        {
        ADCON1 = 0x00 ;          // ADC donusturucu kullanilacak
        PORTA = 0 ;
        TRISA = 0xff ;           // ADC icin PORTA'yı giris olarak kur
        PORTB = 0 ;
        TRISB = 0xff ;           // butonlar icin PORTB'yi giris olarak kur
        PORTD = 0 ;
        TRISD = 0 ;              // PORTD'i cikis olarak kur
        /* ENC28J60'yi sunlarla baslat :
         * reset biti RC0 uzerinde
         * CS biti RC1 uzerinde
         *  benim MAC & IP adreslerim
         * tam cift yonlu haberlesme
         */
Spi_Init();     // SPI modulunu baslat

SPI_Ethernet_Init(&PORTC, 0, &PORTC, 1, myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);
        while(1) // Sonsuza dek asagidakileri islemleri yap
                {
                SPI_Ethernet_doPacket() ; // Ethernet paketlerini isle

                /*
                 * Ihtiyac duyulan maddelerinizi buraya ekleyiniz.
                 * SPI_Ethernet_doPacket() mumkun oldugunca sik cagirilmalidir,
                 * aksi taktirde paketler kaybedilebilir.
                 */
                }
        }
```

Donanım Bağlantısı



Flash Bellek Kütüphanesi

Bu kütüphane mikrodenetleyicinin Flash belleğine erişim için gerekli yordamları sağlar. PIC16 ve PIC18 aileleri için prototipler farklıdır, dikkat ediniz.

Not: PIC16 ailesinin flash özelliklerinden dolayı, flash kütüphanesi MCU'a bağımlıdır. Flash bellek işlemlerini destekleyen üç tip MCU vardır:

1. Sadece flash okuma işlemini desteleyenler. Bu tür MCU grupları için sadece Flash_Read fonksiyonu geçerlidir.
2. Okuma ve yazma işlemlerini destekleyenler (Yazma işlemi silme-yazma olarak çalışmaktadır). Bu tür MCU grupları için yazma ve okuma işlemleri geçerlidir.
3. Okuma, yazma ve silme işlemini destekleyenler. Bu tür MCU grupları için yazma, okuma ve silme işlemleri geçerlidir. Dahası flash bellek bloğu yazma işlemi öncesi silinmelidir (Yazma işlemi silme-yazma olarak çalışmaz).

Flash Kütüphanesini kullanmadan önce MCU'nun özelliklerine bakınız.

Kütüphane Yordamları

```
Flash_Read
Flash_Write
Flash_Erase
```

Flash_Read

Yapısı	`unsigned Flash_Read(unsigned address);` `// PIC16 için` `unsigned short Flash_Read(long address);` `// PIC18 için`
Dönüş	Flash bellekten veri baytını döndürür.
Tanımı	Veriyi belirtilmiş olan Flash bellek adresinden okur.
Örnek	`Flash_Read(0x0D00);`

Flash_Write

Yapısı	`// PIC16 için`
	`void Flash_Write(unsigned address, unsigned int * data);`
	
	`// PIC18 için`
	`void Flash_Write(long address, unsigned short *fdata);`

Tanımı	Veri parçasını Flash belleğe yazar. PIC18 ailesinde yazılacak veri tam tamına 64 bayt uzunluğunda olmalıdır. Akılda tutulması gereken şudur; bu fonksiyon veriyi yazarken önce hedef bellekteki veriyi siler. Bu şu anlama gelmektedir; eğer yazma işlemi başarılı olmadıysa, daha önce kaydedilmiş olan veri kaybedilecektir.

Örnek	`// Ardisik degerleri ardisik 64 bellek yerlesimine yaz`
	`char toWrite[64];`
	
	`// Diziyi doldur:`
	`for (i = 0; i < 63; i++) toWrite[i] = i;`
	`// Flash'a yaz`
	`Flash_Write(0x0D00, toWrite);`

Flash_Erase

Yapısı	`void Flash_Erase(unsigned address);`
Tanımı	Verilen bir adresten başlayarak 32 byte'lık bir veri bloğunu siler. Sadece flash belleği hem yazma hem de silme işlemini gerçekleştiremeyen mikrodenetleyicileri için geçerlidir (Daha fazla bilgi için MCU'nun özelliklerine bakınız).

Örnek	`/* $0D00 adresinden baslayarak 32 baytlik bellek`
	`blogunu sil:*/`
	
	`Flash_Erase($0D00);`

Kütüphane Örneği

Örnek PIC18 için flash belleğe basit bir yazma işlemi gerçekleştirir ve daha sonrasında tekrar bu veriyi okuyup PORTB'de gösterir.

```
unsigned short i = 0, j = 0;
unsigned long addr;
unsigned short dataRd;
unsigned short dataWr[ 64] =
             { 1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
               1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
               1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
               1,2,3,4} ;

void main() {
  PORTB = 0;
  TRISB = 0;
  PORTC = 0;
  TRISC = 0;

  addr = 0x00000A30;           // P18F452 icin gecerli
  Flash_Write(addr, dataWr);

  addr = 0x00000A30;
  for (i = 0; i < 64; i++) {
    dataRd = Flash_Read(addr++);
    PORTB = dataRd;
    Delay_ms(500);
  }

} //~!
```

I2C Kütüphanesi

I²C tam ana MSSP birimi bazı PIC MCU'larda bulunur. mikroC temel I²C modunu destekleyen I²C kütüphanesini bulundurur.

Not: İki I²C modülüne sahip PIC mikrodenetleyicileri, mesela P18F8722, hangi modülü kullanacağınızı belirtmenize gereksinim duyar. Bunu basitçe I2C ye 1 veya 2 numaralarını ekleyerek yapabilirsiniz. Örnek olarak, `I2C2_Wr();` Aynı zamanda, önceki derleyici versiyonlarına uyumlu olması ve basit kod yönetimi için de bunu yapınız. Çoklu I²C modülüne sahip MCUlar, I2C1 ile özdeş I²C kütüphanesine sahiptir. (mesela: `I2C_Init()`'i `I2C1_Init()`'nın yerine kullanabilirsiniz).

Kütüphane Yordamları

```
I2C_Init
I2C_Start
I2C_Repeated_Start
I2C_Is_Idle
I2C_Rd
I2C_Wr
I2C_Stop
```

I2C_Init

Yapısı	`void I2C_Init(long clock);`
Tanımı	I²C'nin başlangıç ayarlarını istenen `clock` ile yapar (Fosc'nin uygun değeri için IC MCU'nun teknik dokümanına bakınız). Bu fonksiyonun; I²C kütüphanesinin diğer yordamlarından önce çağırılması gerekmektedir.
Gereklilikler	Kütüphane PORTB veya PORTC üzerinde MSSP birimi bulunmasını bekler. Not: İki I²C modülü olan MCU'larda ikinci I²C modülünü ayarlamak için kullanacağınız `I2C2_Init` fonksiyonu ise (muhtemelen PORTD 'de) bir MSSP birimi daha bulunmasını bekler.
Örnek	`I2C_Init(100000);`

I2C_Start

Yapısı	`char I2C_Start(void);`
Dönüş	Eğer hiçbir hata yoksa fonksiyon 0 döndürür.
Tanımı	Eğer I²C çoklu hattı serbest ise START sinyalini yayınlar.
Gereklilikler	I²C, bu fonksiyon kullanılmadan önce yapılandırılmalıdır. `I2C_Init`'e bakınız.
Örnek	`I2C_Start();`

I2C_Repeated_Start

Yapısı	`void I2C_Repeated_Start(void);`
Tanımı	Tekrarlı START sinyalini yayınlar.
Gereklilikler	I²C, bu fonksiyon kullanılmadan önce yapılandırılmalıdır. `I2C_Init`'e bakınız.
Örnek	`I2C_Repeated_Start();`

I2C_Is_Idle

Yapısı	`char I2C_Is_Idle(void);`
Dönüş	Eğer I²C çoklu hattı serbest ise 1 döndürür, aksi halde 0 döndürür.
Tanımı	I²C çoklu hattının serbest olup olmadığını test eder.
Gereklilikler	I²C, bu fonksiyon kullanılmadan önce yapılandırılmalıdır. `I2C_Init`'e bakınız.
Örnek	`if (I2C_Is_Idle()) {...}`

I2C_Rd

Yapısı	`char I2C_Rd(char ack);`
Dönüş	Köle olarak kabul edilen taraftan (Slave'den) bir byte döndürür.
Tanımı	Slave'den bir byte okur ve eğer `ack` parametresi 0 (sıfır) ise herhangi bir kabul sinyali yollamaz, aksi halde ise yollar.
Gereklilikler	Bu fonksiyonun kullanılabilmesi için START sinyali yayınlanmış olmalıdır. `I2C_Start`'a bakınız.
Örnek	`temp = I2C_Rd(0); // Veriyi oku ve olumsuz sinyali yolla`

I2C_Wr

Yapısı	`char I2C_Wr(char data);`
Dönüş	Eğer hiçbir hata yoksa fonksiyon 0 döndürür.
Tanımı	Veri byte'ını I²C vasıtasıyla yollar.
Gereklilikler	Bu fonksiyonun kullanılabilmesi için START sinyali yayınlanmış olmalıdır. `I2C_Start`'a bakınız
Örnek	`I2C_Write(0xA3);`

I2C_Stop

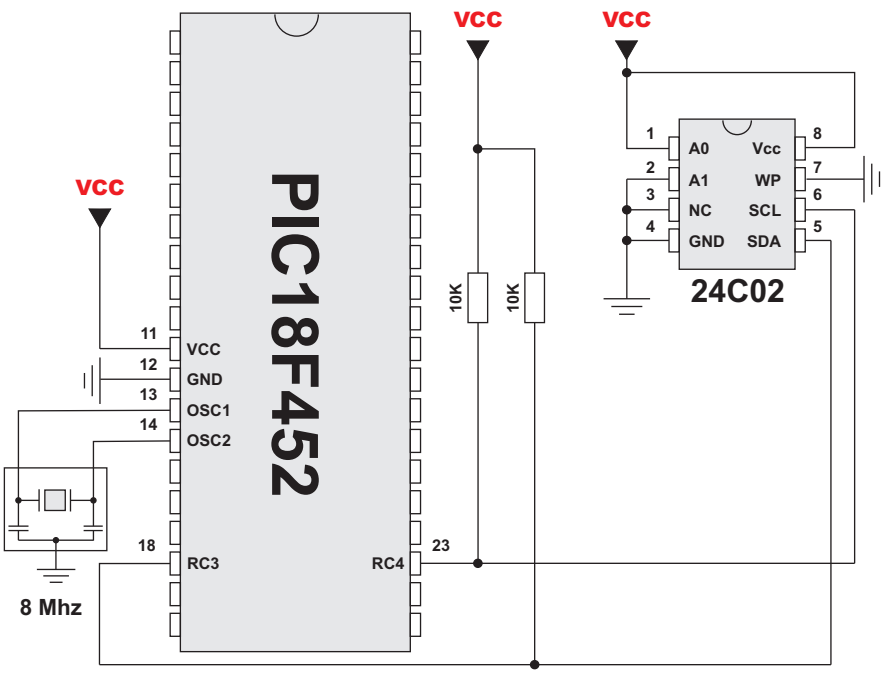
Yapısı	`void I2C_Stop(void);`
Tanımı	STOP sinyali yayınlar.
Gereklilikler	I²C, bu fonksiyon kullanılmadan önce yapılandırılmalıdır. `I2C_Init`'e bakınız.

Kütüphane Örneği

Bu kod parçası I²C kütüphanesinin yordam ve fonksiyonlarınının kullanımını göstermektedir. PIC MCU; SCL, SDA pinleri ile 24c02 EEPROM'a bağlanmıştır. Program veriyi EEPROM'a yollar (veri adres 2'ye yazılır). Daha sonra, veriyi I²C vasıtasıyla EEPROM'dan okur ve PORTB'ye gönderir. Bu işlemin amacı veri aktarım döngüsünün doğru çalışıp çalışmadığını anlamaktır. (Bir sonraki sayfadaki şekil 24c02'den PIC'e arabirim bağlantısını göstermektedir).

```
void main(){
  PORTB = 0;                    // I2C'den alinan verileri gosterecegimiz portu
  TRISB = 0;                    // baslanic durumuna getirelim.

  I2C_Init(100000);             // I2C baslangic ayarlarini yap
  I2C_Start();                  // I2C basla sinyali yayinla
  I2C_Wr(0xA2);                 // I2C yoluyla bayt yolla (24cO2'ye komut)
  I2C_Wr(2);                    // Bayt yolla(EEPROM yerlesim adresi)
  I2C_Wr(0xF0);                 // Veriyi yolla (yazilacak veri)
  I2C_Stop();

  Delay_ms(100);

  I2C_Start();                  // I2C basla sinyali yayınla
  I2C_Wr(0xA2);                 // I2C yoluyla bayt yolla(aygit adresi + W)
  I2C_Wr(2);                    // Bayt yolla (veri adresi)
  I2C_Repeated_Start();         // I2C tekrarli baslama sinyali yayinla
  I2C_Wr(0xA3);                 // Bayt yolla (aygit adresi + R)
  PORTB = I2C_Rd(0u);           // Veriyi oku (onay yok)
  I2C_Stop();
}
```

Donanım Bağlantısı



Tuştakımı Kütüphanesi

mikroC, 4x4 düzenindeki bir tuş takımıyla kullanılmak üzere kütüphane oluşturmuştur. Bu kütüphanenin yordamları 4x1, 4x2 veya 4x3 tuş takımlarıyla da kullanılabilirler. Konunun sonunda verilen donanım bağlantısını dikkatlice inceleyiniz.

Kütüphane Yordamları

```
Keypad_Init
Keypad_Read
Keypad_Released
```

Keypad_Init

Yapısı	`void Keypad_Init(char *port);`
Tanımı	Tuş takımı ile çalışma için `port`'un başlangıç ayarlarını yapar. Yordamın, tuştakımı kütüphanesinden diğer yordamlarının kullanılmasından önce çağırılması gerekir.
Örnek	`Keypad_Init(&PORTB);`

Keypad_Read

Yapısı	`unsigned Keypad_Read(void);`
Dönüş	Basılan tuşa göre 1..16 arasında bir değer döndürür. Hiçbir tuşa basılmazsa 0 döndürür.
Tanımı	Herhangi bir tuşun basılıp basılmadığını kontrol eder. Eğer bir tuşa basılmışsa fonksiyon 1'den 16'ya kadar bir sayı döndürür. Yok eğer hiç bir tuşa basılmamış ise 0 döndürür.
Gereklilikler	Uygun bir şekilde portun başlangıç ayarları yapılmalıdır; `Keypad_Init`'e bakınız.
Örnek	`kp = Keypad_Read();`

Keypad_Released

Yapısı	`unsigned Keypad_Released(void);`
Dönüş	Tuşa bağlı olarak 1'den 16'ya kadar sayıları döndürür.
Tanımı	`Keypad_Released` yordamının çağrılması bir tıkanan (blocking) çağırmadır; fonksiyon bir tuşun basılıp bırakılmasını bekler. Serbest bırakıldığında, fonksiyon basılan tuşa göre 1 ile 16 arasında bir sayı döndürür.
Gereklilikler	Uygun bir şekilde portun başlangıç ayarları yapılmalıdır; `Keypad_Init`'e bakınız.
Örnek	`kp = Keypad_Released();`

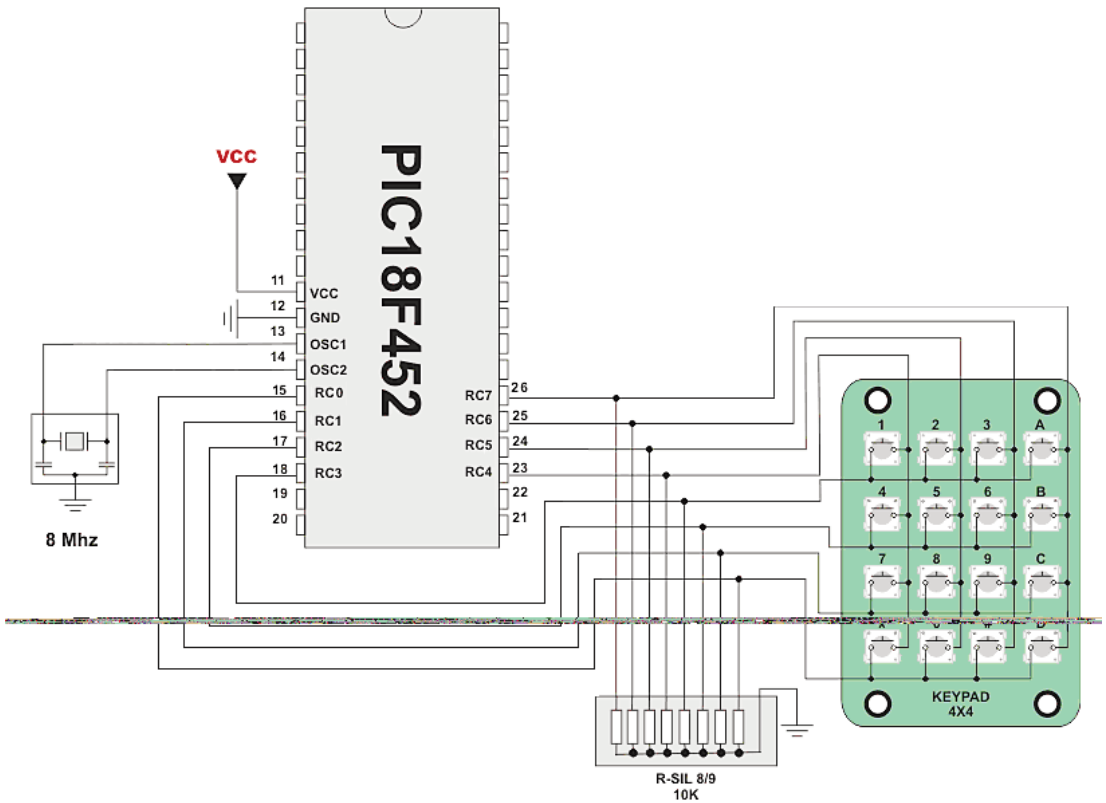
Kütüphane Örneği

Aşağıdaki kod tuş takımını test etmek maksadıyla kullanılabilir. Kütüphane yordamları 1-4 satır ve 1-4 sütun şeklinde bağlanmış tüm tuş takımlarını desteklemektedir. Bu kod, tuş fonksiyonlarından dönen 1-16 sayısını ASCII formatına [0 .. 9, A .. F] çevirir ve LCD'de gösterir.

```
unsigned short kp, cnt, oldstate = 0;
char txt[5];
void main() {
  cnt = 0;
  Keypad_Init(&PORTC);                          // Tus takimini PORTC'ye ayarla
  Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0);      // LCD'yi PORTB'ye ayarla
  Lcd_Cmd(LCD_CLEAR);                           // Ekrani temizle
  Lcd_Cmd(LCD_CURSOR_OFF);                      // Imleci gizle
  Lcd_Out(1, 1, "Key  :");
  Lcd_Out(2, 1, "Times:");

  do {
    kp = 0;
    // Bir tusun basilmasini bekle
    do
      //kp = Keypad_Released();
      kp = Keypad_Read();
    while (!kp);

    // Degeri cikis icin hazirla
    switch (kp) {
//      case 10: kp = 42; break;  // '*'
//      case 11: kp = 48; break;  // '0' // keypad4x3 icin bu alani aciniz
//      case 12: kp = 35; break;  // '#'
//      default: kp += 48;
      case  1: kp = 49; break; // 1
      case  2: kp = 50; break; // 2
      case  3: kp = 51; break; // 3
      case  4: kp = 65; break; // A
      case  5: kp = 52; break; // 4
      case  6: kp = 53; break; // 5
      case  7: kp = 54; break; // 6
      case  8: kp = 66; break; // B  // keypad4x4 icin bu alani aciniz
      case  9: kp = 55; break; // 7
      case 10: kp = 56; break; // 8
      case 11: kp = 57; break; // 9
      case 12: kp = 67; break; // C
      case 13: kp = 42; break; // *
      case 14: kp = 48; break; // 0
      case 15: kp = 35; break; // #
      case 16: kp = 68; break; // D
    }
```

```
      if (kp != oldstate) {
        cnt = 1;
        oldstate = kp;
        while(!Keypad_Released()) asm nop;
        }
      else {
        cnt++;
        while(!Keypad_Released()) asm nop;
        }
      Lcd_Chr(1, 10, kp);  // LCD'ye yaz
      if (cnt == 255) {
        cnt = 0;
        Lcd_Out(2, 10, "   ");
      }
      WordToStr(cnt, txt);
      Lcd_Out(2, 10, txt);
    } while (1);
}
```

Donanım Bağlantısı



LCD Kütüphanesi (4-bit arabirimli)

mikroC genel kullanımlı LCD'ler ile haberleşme için (4-bit arabirimli) bir kütüphane oluşturmuştur. PIC'in ve LCD'nin donanım bağlantısı bölüm sonunda gösterilmiştir.

Not: Aşağıdaki kütüphane yordamlarından birini kullanmadan önce LCD'nin bağlandığı portun çıkış olarak ayarlanmış olmasına dikkat ediniz.

Kütüphane Yordamları

```
Lcd_Config
Lcd_Init
Lcd_Out
Lcd_Out_Cp
Lcd_Chr
Lcd_Chr_Cp
Lcd_Cmd
```

Lcd_Config

Yapısı	`void Lcd_Config(char *port, char RS, char EN, char WR, char D7, char D6, char D5, char D4);`
Tanımı	Kullanıcının belirttiği pin ayarları ile LCD portunu başlangıç durumuna getirir: RS, EN, WR, D7 .. D4 parametreleri 0'dan 7'ye kadar olan değerlerin bir kombinasyonu olmalıdır (Örneğin: 3,6,0,7,2,1,4).

Örnek	`// Ilgili Port, LCD pinleri ve karsilik gelen MCU pinleri :`
	`//`
	`//  port  RS EN WR D7 D6 D5 D4`
	`Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0); // EasyPIC5 icin ayar.`

Lcd_Init

Yapısı	`void Lcd_Init(char *port);`

Tanımı	Varsayılan pin ayarları ile LCD portunu başlangıç durumuna getirir. (bölüm sonundaki bağlantı şemasına bakınız):
	`D7 -> PORT.7,`
	`D6 -> PORT.6,`
	`D5 -> PORT.5,`
	`D4 -> PORT.4,`
	`E -> PORT.3,`
	`RS -> PORT.2`
	
	Not: EasyPIC5 kartı için geçerli pin ayarları için bu yordam yerine `Lcd_Config` yordamını kullanınız.

Örnek	`Lcd_Init(&PORTB);`

Lcd_Out

Yapısı	`void Lcd_Out(char row, char col, char *text);`
Tanımı	Metni (`text`) LCD'nin belirtilen satır ve kolonuna (`row` ve `col` parametreleri) yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Config` veya `Lcd_Init`'e bakınız.
Örnek	`Lcd_Out(1, 3, "merhaba!"); /* "merhaba!" yazisini 1 satir 3. karakterden itibaren yaz.*/`

Lcd_Out_Cp

Yapısı	`void Lcd_Out_Cp(char *text);`
Tanımı	Metni (`text`) LCD'ye; imleç pozisyondan başlayarak yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Config` veya `Lcd_Init`'e bakınız.
Örnek	`Lcd_Out_Cp("buradayim!"); /* Su anki imlec pozisyonuna "buradayim!" yazisini yaz. */`

Lcd_Chr

Yapısı	`void Lcd_Chr(char row, char col, char character);`
Tanımı	Karakteri (`character`) LCD'nin belirtilen satır ve kolonuna (`row ve col` parametreleri) yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Config` veya `Lcd_Init`'e bakınız.
Örnek	`Lcd_Out(2, 3, 'i');  /* "i" yi 2.satir 3. karakterden baslayarak yaz*/`

Lcd_Chr_Cp

Yapısı	`void Lcd_Chr_Cp(char character);`
Tanımı	Karakteri (`character`) LCD'ye; imleç pozisyondan başlayarak yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Config` veya `Lcd_Init`'e bakınız.
Örnek	`Lcd_Chr_Cp('e');  // "e" yi imlecin bulundugu yere yaz`

Lcd_Cmd

Yapısı	`void Lcd_Cmd(char command);`
Tanımı	Komutu (`command`) LCD'ye yollar. Fonksiyona önceden belirtilmiş sabitlerden birini geçebilirsiniz. Geçerli tüm komutlar bir sonraki sayfada verilmiştir.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Config` veya `Lcd_Init`'e bakınız.
Örnek	`Lcd_Cmd(LCD_CLEAR);  // LCD ekranini temizle`

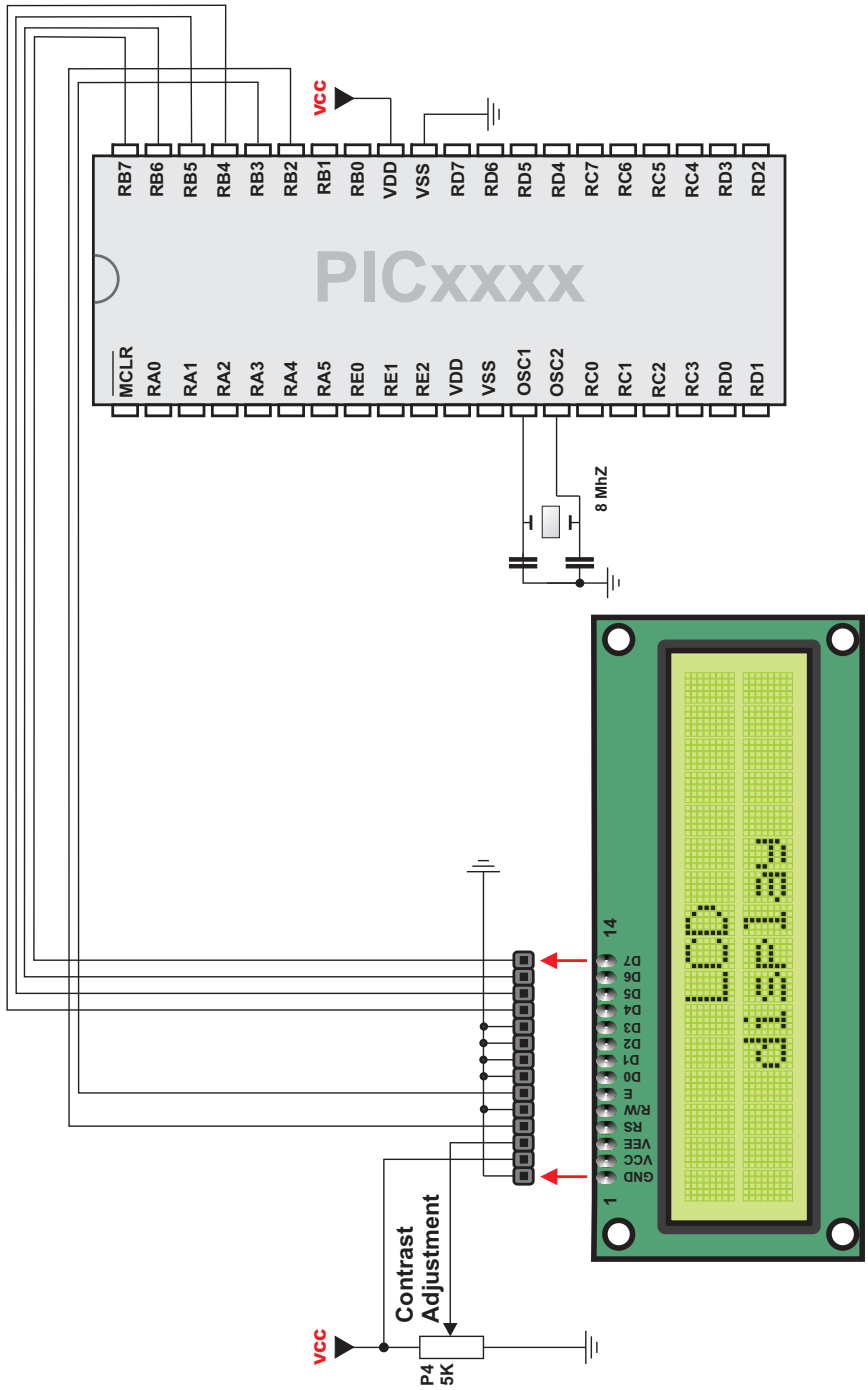
LCD Komutları

LCD Komutu	Amacı
LCD_FIRST_ROW	İmleci birinci satıra taşır
LCD_SECOND_ROW	İmleci ikinci satıra taşır
LCD_THIRD_ROW	İmleci üçüncü satıra taşır
LCD_FOURTH_ROW	İmleci dördüncü satıra taşır
LCD_CLEAR	Ekranı temizler
LCD_RETURN_HOME	İmleci başlangıç pozisyonuna getirir, kaydırılmış bir görüntüyü eski durumuna getirir. “Görüntü veri RAM”’ının içeriği değişmez.
LCD_CURSOR_OFF	İmleci kapatır
LCD_UNDERLINE_ON	İmlecin altını çizme işlemini açar
LCD_BLINK_CURSOR_ON	İmlecin yanıp-sönme (Blink) işlemini açar
LCD_MOVE_CURSOR_LEFT	Görüntü veri RAM’ının içeriğini değiştirmeden imleci sola taşır
Move_CURSOR_RIGHT	Görüntü veri RAM’ının içeriğini değiştirmeden imleci sağa taşır
LCD_TURN_ON	LCD ekranını açar
LCD_TURN_OFF	LCD ekranını kapatır
LCD_SHIFT_LEFT	Görüntü veri RAM’ının içeriğini değiştirmeden veriyi sola kaydırır
LCD_SHIFT_RIGHT	Görüntü veri RAM’ının içeriğini değiştirmeden veriyi sağa kaydırır

Kütüphane Örneği (geçerli pin ayarları)

```
char *text = "beti bilisim";

void main() {
  TRISB = 0;                      // PORTB cikis
  Lcd_Init(&PORTB);               // PORTB'ye bagli LCD'yi varsayilan ayara kur.
  // Lcd_Config(&PORTB, 4, 5, 6, 3, 2, 1, 0); // EasyPIC5 karti icin ozel ayar.

  Lcd_Cmd(Lcd_CLEAR);             // Ekrani temizle
  Lcd_Cmd(Lcd_CURSOR_OFF);        // Imleci kapat
  Lcd_Out(1, 1, text);            // Metni LCD'ye yaz, 1. satir, 1. sutundan
}//~!
```

Hardware Connection



Özel LCD kütüphanesi (4-bit arabirimli)

mikroC genel kullanımlı "fakat pinlerinin port içindeki yerleşim sırası özel" olan LCD'ler ile haberleşme için (4-bit arabirimli) bir kütüphane oluşturmuştur. PIC'in ve LCD'nin donanım bağlantısı bölüm sonunda gösterilmiştir.

Kütüphane Yordamları

```
Lcd_Custom_Config
Lcd_Custom_Out
Lcd_Custom_Out_Cp
Lcd_Custom_Chr
Lcd_Custom_Chr_Cp
Lcd_Custom_Cmd
```

Lcd_Custom_Config

Yapısı	`void Lcd_Custom_Config(char * data_port, char D7, char D6, char D5, char D4, char * ctrl_port, char RS, char WR, char EN);`
Tanımı	LCD data ve kontrol pinlerini seçtiğiniz pin ayarları ile başlangıç durumuna getirir.

Örnek	`// Ornek1`
	`Lcd_Custom_Config(&PORTD,3,2,1,0,&PORTB,2,3,4);`
	
	`// EasyPIC5 karti icin ayarlar.`
	`// Ilgili Port, LCD pinleri ve karsilik gelen MCU pinleri :`
	`//`
	`// data_port, D7, D6, D5, D4, ctrl_port, RS, WR, EN`
	`Lcd_Custom_Config(&PORTB,3,2,1,0,&PORTB,4,6,5);`

Lcd_Custom_Out

Yapısı	`void Lcd_Custom_Out(char row, char col, char *text);`
Tanımı	Metni (`text`) LCD'nin belirtilen kolon veya satırına (`row` ve `col` parametreleri) yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Custom_Config`'e bakınız.
Örnek	`Lcd_Custom_Out(1, 3, "merhaba!");/* "merhaba!" yazisini 1. satir 3. karakterden baslayarak yaz: */`

Lcd_Custom_Out_Cp

Yapısı	`void Lcd_Custom_Out_Cp(char *text);`
Tanımı	Metni (`text`) LCD'ye mevcut imleç pozisyonundan başlayarak yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Custom_Config`'e bakınız.
Örnek	`Lcd_Custom_Out_Cp("buradayim!"); /* Su anki imlec pozisyonuna "buradayim!" yazisini yaz. */`

Lcd_Custom_Chr

Yapısı	`void Lcd_Custom_Chr(char row, char col, char character);`
Tanımı	Karakteri (`character`) LCD'nin belirtilen kolon veya satırına (`row ve col` parametreleri) yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Custom_Config`'e bakınız.
Örnek	`Lcd_Custom_Chr(2, 3, 'i'); /* "i" harfini LCD 2. satır ve 3. karaktere yaz: */`

Lcd_Custom_Chr_Cp

Yapısı	`void Lcd_Custom_Chr_Cp(char character);`
Tanımı	Karakteri (`character`) mevcut imleç pozisyonundan başlayarak LCD'ye yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Custom_Config`'e bakınız.
Örnek	`Lcd_Custom_Chr_Cp('e'); /* "e" harfini mevcut imlec pozisyonuna yaz: */`

Lcd_Custom_Cmd

Yapısı	`void Lcd_Custom_Cmd(char command);`
Tanımı	Komutu LCD'ye yollar. Fonksiyona önceden belirtilmiş sabitlerden birini geçebilirsiniz. Mevcut tüm komutların listesi bir sonraki sayfada verilmiştir.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd_Custom_Config`'e bakınız.
Örnek	`Lcd_Custom_Cmd(LCD_CLEAR); // LCD ekranini temizle`

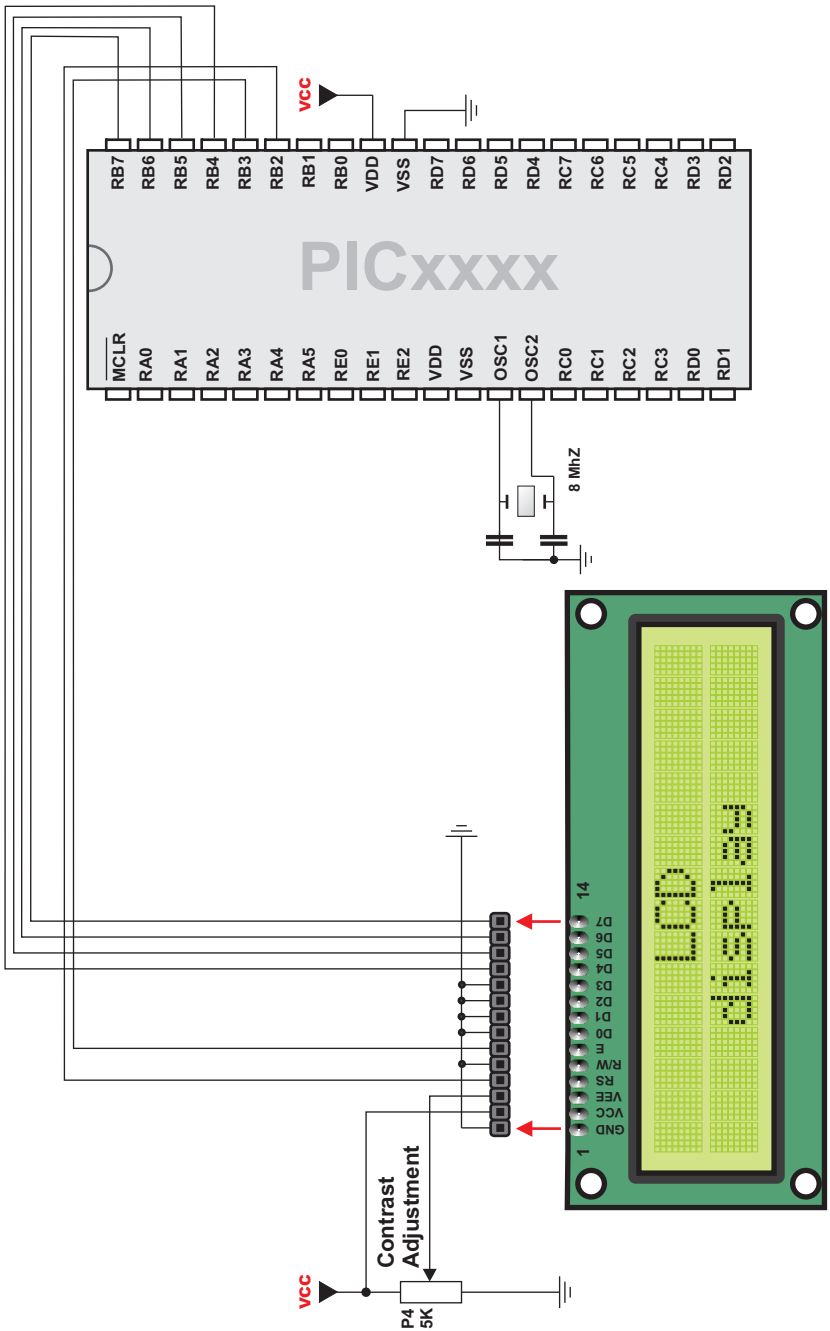
LCD Komutları

LCD Komutu	Amacı
LCD_FIRST_ROW	İmleci birinci satıra taşır
LCD_SECOND_ROW	İmleci ikinci satıra taşır
LCD_THIRD_ROW	İmleci üçüncü satıra taşır
LCD_FOURTH_ROW	İmleci dördüncü satıra taşır
LCD_CLEAR	Ekranı temizler
LCD_RETURN_HOME	İmleci başlangıç pozisyonuna getirir, kaydırılmış bir görüntüyü eski durumuna getirir. Görüntü veri RAM'ının içeriği değişmez.
LCD_CURSOR_OFF	İmleci kapatır
LCD_UNDERLINE_ON	İmlecin altını çizme işlemini açar
LCD_BLINK_CURSOR_ON	İmlecin yanıp-sönme (Blink) işlemini açar
LCD_MOVE_CURSOR_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sola taşır
LCD_MOVE_CURSOR_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sağa taşır
LCD_TURN_ON	LCD ekranını açar
LCD_TURN_OFF	LCD ekranını kapatır
LCD_SHIFT_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sola kaydırır
LCD_SHIFT_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sağa kaydırır

Kütüphane Örneği (varsayılan pin ayarları)

```
char *text = "beti";

void main() {

  TRISB = 0;                                // PORTB cikis
  Lcd_Custom_Config(&PORTB,3,2,1,0,&PORTB,4,6,5); /* PORTB'ye bagli
                                                  LCD'yi PORTB baslangic
                                                  durumuna getir */
  Lcd_Custom_Cmd(Lcd_CLEAR);                // ekrani temizle
  Lcd_Custom_Cmd(Lcd_CURSOR_OFF);           // imleci kapat
  Lcd_Custom_Out(1, 1, text);               /* metni LCD ekranina,
                                               1.satir 1. sutundan itibaren yaz */

} //~!
```

Donanım Bağlantısı



LCD Kütüphanesi (8-bit arabirimli)

mikroC genel kullanımlı, 8-bit arabirimli LCD'ler (Hitachi HD44780 denetleyicili) ile haberleşme için bir kütüphane oluşturmuştur. PIC'in ve LCD'nin donanım bağlantısı bölüm sonunda gösterilmiştir.

Kütüphane Yordamları

```
Lcd8_Config
Lcd8_Init
Lcd8_Out
Lcd8_Out_Cp
Lcd8_Chr
Lcd8_Chr_Cp
Lcd8_Cmd
```

Lcd8_Config

Yapısı	`void Lcd8_Config(char *ctrlport, char *dataport, char RS, char EN, char WR, char D7, char D6, char D5, char D4, char D3, char D2, char D1, char D0);`
Tanımı	Kontrol portu (`ctrlport`) veri portu (`dataport`) seçimi ile LCD'yi başlangıç durumuna getirir: RS, EN, WR parametreleri 0 ile 7 arasında olmalıdır. D7 .. D0 parametreleri ise 0'dan 7'ye kadar olan değerlerin bir kombinasyonu olmalıdır. (Örneğin : 3,6,0,7,2,1,4).
Örnek	`Lcd8_Config(&PORTC,&PORTD,0,1,2,6,5,4,3,7,1,2,0);`

Lcd8_Init

Yapısı	`void Lcd8_Init(char *ctrlport, char *dataport);`
Tanımı	Kontrol portu (`ctrlport`) veri portu (`dataport`) seçimi ile LCD'yi "varsayılan pin ayarlarına sadık kalarak" başlangıç durumuna getirir: (bölüm sonundaki bağlantı şemasına bakınız). Varsayılan pin ayarları şunlardır: E -> ctrlport.3, RS -> ctrlport.2, R/W -> ctrlport.0, D7 -> dataport.7, D6 -> dataport.6, D5 -> dataport.5, D4 -> dataport.4, D3 -> dataport.3, D2 -> dataport.2, D1 -> dataport.1, D0 -> dataport.0
Örnek	`Lcd8_Init(&PORTB, &PORTC);`

Lcd8_Out

Yapısı	`void Lcd8_Out(char row, char col, char *text);`
Tanımı	Metni (`text`) LCD'nin belirtilen kolon veya satırına (`row` ve `col` parametreleri) yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd8_Config` veya `Lcd8_Init`'e bakınız.
Örnek	`Lcd8_Out(1, 3, "merhaba"); /* 1. satir ve 3. karakterden baslayarak "merhaba" yaz */`

Lcd8_Out_Cp

Yapısı	`void Lcd8_Out_Cp(char *text);`
Tanımı	Metni (`text`) LCD'ye mevcut imleç pozisyonundan başlayarak yazar. Hem karakter dizileri hem de karakter dizi değişmezleri `text` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd8_Config` veya `Lcd8_Init`'e bakınız.
Örnek	`Lcd8_Out_Cp("buradayim!"); /* Su anki imlec pozisyonuna "buradayim!" yazisini yaz. */`

Lcd8_Chr

Yapısı	`void Lcd8_Chr(char row, char col, char character);`
Tanımı	Karakteri (`character`) LCD'nin belirtilen kolon veya satırına (`row ve col` parametreleri) yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd8_Config` veya `Lcd8_Init`'e bakınız.
Örnek	`Lcd8_Out(2, 3, 'i'); // "i" harfini 2. satir 3. karaktere yaz`

Lcd8_Chr_Cp

Yapısı	`void Lcd8_Chr_Cp(char character);`
Tanımı	Karakteri (`character`) mevcut imleç pozisyonundan başlayarak LCD'ye yazar. Hem değişkenler hem de değişmezler `character` olarak geçirilebilirler.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd8_Config` veya `Lcd8_Init`'e bakınız.
Örnek	`Lcd8_Chr_Cp('e'); /* "e" harfini mevcut imlec pozisyonuna yaz: */`

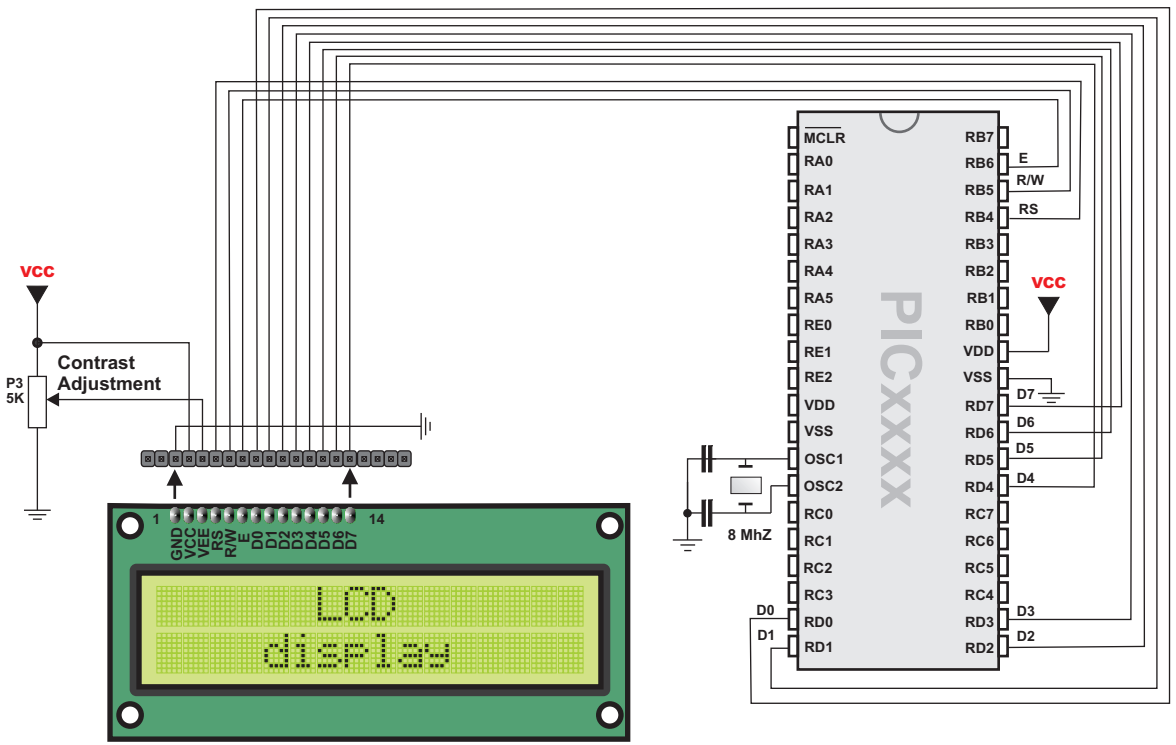
Lcd8_Cmd

Yapısı	`void Lcd8_Cmd(char command);`
Tanımı	Komutu LCD'ye yollar. Fonksiyona önceden belirtilmiş sabitlerden birini geçebilirsiniz. Mevcut tüm komutların listesi önceki sayfalarda verilmiştir.
Gereklilikler	LCD portu başlangıç durumuna getirilmiş olmalıdır. `Lcd8_Config` veya `Lcd8_Init`'e bakınız.
Örnek	`Lcd8_Cmd(Lcd_Clear); // LCD ekranini temizle`

KÜtüphane Örneği (varsayılan pin ayarları ile)

```
char *text = "beti";

void main() {
  TRISB = 0;                    // PORTB cikis
  TRISD = 0;                    // PORTD cikis
  Lcd8_Init(&PORTB, &PORTD);    // LCD'yi PORTB ve PORTD'ye ayarla
  Lcd8_Cmd(Lcd_CURSOR_OFF);     // Imleci gizle
  Lcd8_Out(1, 1, text);         // Metni LCD'ye yaz
}
```

Donanım Bağlantısı



Grafik LCD (GLCD) Kütüphanesi

mikroC, Grafik LCD üzerinde yazma ve çizme için bir kütüphane sağlar. Bu yordamlar sadece genel kullanımlı 128x64 GLCD'leri ve PIC18 ailesi ile kullanılmaktadır.

Kütüphane Yordamları

Temel Yordamlar:

```
Glcd_Init
Glcd_Set_Side
Glcd_Set_Page
Glcd_Set_X
Glcd_Read_Data
Glcd_Write_Data
```

Gelişmiş Yordamlar:

```
Glcd_Fill
Glcd_Dot
Glcd_Line
Glcd_V_Line
Glcd_H_Line
Glcd_Rectangle
Glcd_Box
Glcd_Circle
Glcd_Set_Font
Glcd_Write_Char
Glcd_Write_Text
Glcd_Image
```

Glcd_Init

Yapısı	`void Glcd_Init(unsigned char *ctrl_port, char cs1, char cs2, char rs, char rw, char rst, char en, unsigned char *data_port);`
Tanımı	GLCD'yi belirttiğiniz pin numarası ayarları ve veri port (`data_port`)'u ile başlangıç durumuna getirir. `cs1, cs2, rs, rw, rst ve en` pinleri herhangi boş bir portun pinleri olabilir. Bu yordam; GLCD kütüphanesinin diğer yordamlarının kullanımından önce çağırılmalıdır.

Örnek	`// EasyPIC5 gelistirme karti icin alisilagelmis ayar`
	`Glcd_Init(&PORTB,  0, 1, 2, 3, 5, 4, &PORTD);`
	
	`// EasyPIC5 icin, cs1 ve cs2'leri tam ters olan GLCD'lerin`
	`// kullanimi istendiginde onerilen ayar`
	`Glcd_Init(&PORTB,  1, 0, 2, 3, 5, 4, &PORTD);`

Glcd_Set_Side

Yapısı	`void Glcd_Set_Side(unsigned short x);`
Tanımı	LCD'nin sağ veya sol tarafını seçili duruma getirir. `x` parametresi seçilecek tarafı gösterir. 0'dan 63'e kadar değerler sol taraf ve 64'ün üzerindekiler ise sağ tarafı belirler. `Glcd_Set_Side, Glcd_Set_X ve Glcd_Set_Page` fonksiyonları ile GLCD üzerinde tam bir pozisyon (yerleşim) belirleyebilirsiniz. Daha sonra ise `Glcd_Write_Data` veya `Glcd_Read_Data` komutlarını ilgili pozisyona yazma ve okuma işlemleri için kullanabilirsiniz.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Select_Side(0);`

Glcd_Set_Page

Yapısı	`void Glcd_Set_Page(unsigned short page);`
Tanımı	GLCD'nin sayfasını (yani teknik olarak satırını) seçilmiş duruma getirir. `page` parametresi yani satır numarası 0'dan 7'ye kadar bir sayı olabilir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Set_Page(5);  // 5'inci satirda konumlan`

Glcd_Set_X

Yapısı	`void Glcd_Set_X(unsigned short x_pos);`
Tanımı	Verilen sayfada (yani satırda) GLCD nin sol sınırından itibaren `x` nokta sonrasında yerleşim alır. (Tabi ki halen seçili olan GLCD tarafı da önemlidir).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Set_X(25);  // Soldan 25'inci noktada konumlan.`

Glcd_Read_Data

Yapısı	`unsigned short Glcd_Read_Data(void);`
Dönüş	GLCD hafızasından bir kelime (word) okur.
Tanımı	Veriyi GLCD belleğinin mevcut bölgesinden okur. `Glcd_Set_Side`, `Glcd_Set_X` ve `Glcd_Set_Page` fonksiyonları ile GLCD üzerinde tam bir konum belirleyebilirsiniz. Daha sonra `Glcd_Write_Data` veya `Glcd_Read_Data` komutlarını bu yerleşim üzerine yazma ve okuma için kullanabilirsiniz.
Gereklilikler	Veriyi GLCD belleğinin mevcut bölgesinden okur.
Örnek	`tmp = Glcd_Read_Data();`

Glcd_Write_Data

Yapısı	`void Glcd_Write_Data(unsigned short data);`
Tanımı	Veriyi (`data`) GLCD belleğinin geçerli bölgesine yazar ve bir sonraki bölgeye hareket eder. Not: Her yazma bölgesi ilgili satırda "üst üste noktaların oluşturduğu dikey bir cizgi parçasi"dır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Write_Data(data);`

Glcd_Fill

Yapısı	`void Glcd_Fill(unsigned short pattern);`
Tanımı	GLCD belleğini byte örnekleri ile doldurur. GLCD ekranını temizlemek için `Glcd_Fill(0)` komutu kullanılır. GLCD ekranını tümüyle doldurmak için `Glcd_Fill($FF)` komutu kullanılır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Fill(0);  // Ekranı temizle`

Glcd_Dot

Yapısı	`void Glcd_Dot(unsigned short x, unsigned short y, char color);`
Tanımı	GLCD üzerinde (`x, y`) koordinatları ile belirlenen yere bir nokta koyar. Renk (`color`) parametresi noktaların durumunu belirtir; 0: nokta yok, 1: nokta var, ve 2: noktanın varlık (renk) durumunu tersine çevirir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Dot(0, 0, 2); // Sol ust kosedeki noktayi tersle`

Glcd_Line

Yapısı	`void Glcd_Line(int x1, int y1, int x2, int y2, char color);`
Tanımı	GLCD üzerinde (`x1,y1`)'den (`x2,y2`) komununa kadar bir çizgi çizer. Renk (`color`) parametresi noktaların durumunu belirtir; 0 :boş çizgi, 1:dolu çizgi ve 2 : çizginin her noktasını tersine çevirir (ve şık bir çizgi çizer).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Line(0, 63, 50, 0, 2);`

Glcd_V_Line

Yapısı	`void Glcd_V_Line(unsigned short y1, unsigned short y2, unsigned short x, char color);`
Tanımı	`Glcd_Line`'a benzer, olarak (`x, y1`)'den (`x,y2`)'ye dikey bir çizgi çizer.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_V_Line(0, 63, 0, 1);`

Glcd_H_Line

Yapısı	`void Glcd_H_Line(unsigned short x1, unsigned short x2, unsigned short y, char color);`
Tanımı	`Glcd_Line`'a benzer, olarak (`x1,y`)'den (`x2,y`)'ye yatay bir çizgi çizer.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_H_Line(0, 127, 0, 1);`

Glcd_Rectangle

Yapısı	`void Glcd_Rectangle(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, char color);`
Tanımı	GLCD üzerine bir dikdörtgen çizer. (`x1,y1`) parametreleri sol üst köşeyi, (`x2,y2`) sağ alt köşeyi ayarlar. Renk parametresi (`color`) kenar sınırları belirler: 0 :boş kenar (kenarlarda nokta yok), 1:dolu kenar (kenarlarda nokta var) ve 2 : akıllı (smart) sınır (kenarlarda her noktayı tersine çevirir).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Rectangle(10, 0, 30, 35, 1);`

Glcd_Box

Yapısı	`void Glcd_Box(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, char color);`
Tanımı	GLCD üzerine bir kutu çizer. (`x1,y1`) paramatreleri sol üst köşeyi, (`x2,y2`) sağ alt köşeyi ayarlar. Renk parametresi (`color`) kutuyu doldurmayı belirler: 0:beyaz kutu (içine nokta koyulmaz), 1:dolu kutu (içine nokta koyulur) ve 2 : tersine dönmüş kutu (her noktayı tersine çevirir).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Box(10, 0, 30, 35, 1);`

Glcd_Circle

Yapısı	`void Glcd_Circle(int x, int y, int radius, char color);`
Tanımı	GLCD üzerine (`x,y`) merkezli bir çember çizer. Renk parametresi (`color`) çember çizgilerini belirler: 0 :boş çember (içinde nokta yok), 1:dolu çember (içinde nokta var) ve 2 : akıllı (smart) çember (her noktayı tersine çevirir).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Circle(63, 31, 25, 2);`

Glcd_Set_Font

Yapısı	`void Glcd_Set_Font(const char *font, unsigned short font_width, unsigned short font_height, unsigned font_offset);`
Tanımı	`Glcd_Write_Char` ve `Glcd_Write_Text` yordamları için kullanılacak font'u ayarlar. `Font`'un bir bayt dizisi olarak düzenlenmesi gerekir. `font_width ve font_height` parametreleri karakterin nokta cinsinden genişliğini ve yüksekliğini gösterir. Font genişliği 128 noktayı geçmemeli, fontun yüksekliği de 8 noktayı geçmemelidir. `font_offset` parametresi fontun hangi ASCII karakterden başladığını belirler. Kütüphane içinde verilmiş demo fontlar 32 offsete sahiptirler, dolayısıyla da boşluk (space) ile başlarlar. Eğer hiçbir font belirtilmemiş ise; `Glcd_Write_Char` ve `Glcd_Write_Text` yordamları kütüphane ile verilmiş olan 5x8 font'u kullanırlar. "GLCD_fonts.c" dosyasında verilmiş rehber bilgilerle kendi fontunuzu oluşturabilirsiniz. Bu dosya GLCD için geçerli fontları içerir ve kurulum dizinine yerleştirilmiştir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`// space(32)ile baslayan ozel 5x7 font'u ("myfont") kullan:` `Glcd_Set_Font(myfont_5x8, 5, 8, 32);`

Glcd_Write_Char

Yapısı	`void Glcd_Write_Char(unsigned short character, unsigned short x, unsigned short page, char color);`
Tanımı	Sayfaya (`page`) (yani 8 GLCD satırından biri, 0..7) karakteri (`character`) yazar. `x` parametresi karakterin ekranın sol kenarına olan nokta uzaklığını belirtir. Renk parametresi (`color`) doldurmayı belirler: 0:beyaz karakter (nokta yok), 1:dolu karakter (nokta var) ve 2 : akıllı (smart) karakter (her noktayı tersine çevirir). Öncesinde `Glcd_Set_Font` yordamını kullanarak geçerli font'u belirleyiniz. Bir font belirlemezseniz kütüphane ile verilen 5x7'lik font geçerli olacaktır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Write_Char('C', 0, 0, 1);`

Glcd_Write_Text

Yapısı	`void Glcd_Write_Text(char *text, unsigned short x, unsigned short page, unsigned short color);`
Tanımı	Sayfaya (`page`) (yani 8 GLCD satırından biri, 0..7) metni (`text`) yazar. `x` parametresi karakterin ekranın sol kenarına olan nokta uzaklığını belirtir. Renk (`color`) parametresi noktaların durumunu belirtir; 0:beyaz karakter (nokta yok) , 1:dolu karakter (nokta var) ve 2: akıllı karakter (her nokta tersine çevrilmiştir).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Write_Text("merhaba!", 0, 0, 1);`

Glcd_Image

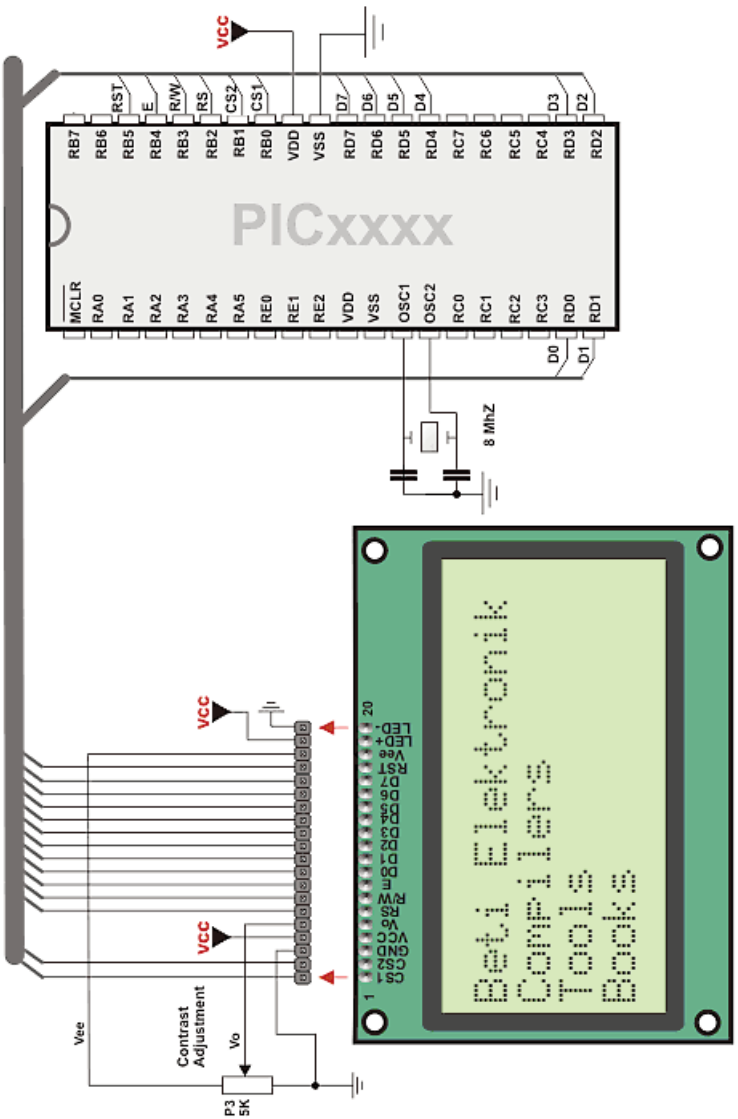
Yapısı	`void Glcd_Image(const char *image);`
Tanımı	Bit-harita (bitmap) görüntüsünü GLCD üzerinde gösterir. `image` parametresi 1024 byte'lık bir dizi gibi biçimlendirilmelidir. Not: mikroC kendi içerisinde Bitmapden LCD biçimine bir çevirme işlemi için Bitmap-to-LCD editörünü bulundurmaktadır (Ana Menü>tools>GLCD Bitmap Editor)
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Glcd_Init`'e bakınız.
Örnek	`Glcd_Image(my_image);`

Kütüphane Örneği

Aşağıdaki demo çizim GLCD kütüphanesinin gelişmiş yordamlarını test etmektedir.

```
unsigned short j, k;
void main() {
  // ctrl_port, cs1, cs2, rs, rw, rst, en, data_port -> LCD tarafindaki pinler
  Glcd_Init(&PORTB,  0,  1,  2, 3, 5,  4, &PORTD); // EasyPIC5 icin normal ayar
  // EasyPIC5 deneme karti icin chip selecti farkli LCD'ler icin onerilen ayar
  // Glcd_Init(&PORTB,  1,  0,  2, 3, 5,  4, &PORTD);

  // Ekran yazilari icin font'u ayarla
  Glcd_Set_Font(FontSystem5x8, 5, 8, 32);

  do {
    // Cemberler ciz
    Glcd_Fill(0);  // Ekrani temizle
    Glcd_Write_Text("Circles", 0, 0, 1);
    j = 4;
    while (j < 31) {
      Glcd_Circle(63, 31, j, 2);
      j += 4;
    }
    Delay_ms(4000);

    // Kutular ciz
    Glcd_Fill(0);  // Ekrani temizle
    Glcd_Write_Text("Rectangles", 0, 0, 1);
    j = 0;
    while (j < 31) {
      Glcd_Box(j, 0, j + 20, j + 25, 2);
      j += 4;
    }
    Delay_ms(4000);

    // Cizgiler ciz
    Glcd_Fill(0);  // Ekrani temizle
    Glcd_Write_Text("Lines", 0, 0, 1);
    for (j = 0; j < 16; j++) {
      k = j*4 + 3;
      Glcd_Line(0, 0, 127, k, 2);
    }
    for (j = 0; j < 31; j++) {
      k = j*4 + 3;
      Glcd_Line(0, 63, k, 0, 2);
    }
    Delay_ms(4000);
  } while (1);
} //~!
```

Donanım Bağlantısı



Toshiba T6963C Grafik LCD Kütüphanesi

mikroC Toshiba T6963C Grafik LCD üzerinde yazma ve çizme için bir kütüphane sağlar (büyüklük değiştirilebilir).

Kütüphane Yordamları

```
T6963C_Init
T6963C_writeData
T6963C_writeCommand
T6963C_setPtr
T6963C_waitReady
T6963C_fill
T6963C_dot
T6963C_write_char
T6963C_write_text
T6963C_line
T6963C_rectangle
T6963C_box
T6963C_circle
T6963C_image
T6963C_sprite
T6963C_set_cursor
T6963C_clearBit
T6963C_setBit
T6963C_negBit
T6963C_displayGrPanel
T6963C_displayTxtPanel
T6963C_setGrPanel
T6963C_setTxtPanel
T6963C_panelFill
T6963C_grFill
T6963C_txtFill
T6963C_cursor_height
T6963C_graphics
T6963C_text
T6963C_cursor
T6963C_cursor_blink
T6963C_Init_240x128
T6963C_Init_240x64
```

T6963C_init

Yapısı	`void T6963C_init(unsigned int w, unsigned int h, unsigned int fntW, unsigned int *data, unsigned int *cntrl, unsigned int bitwr, unsigned int bitrd, unsigned int bitcd, unsigned int bitreset);`
Tanımı	Grafik LCD denetleyicisini başlangıç durumuna getirir. Bu yordam; T6963C kütüphanesinin diğer yordamlarının kullanımından önce çağırılmalıdır. `width` – görüntü birimindeki (x) yatay nokta sayısı. `height` – görüntü birimindeki (y) düşey nokta sayısı. `fntW`– font genişliği, text karakterindeki nokta sayısı donanıma göre belirlenmelidir. `data` – veri hatlarının bağlandığı port'un adresi. `cntrl` – kontrol hatlarının bağlandığı port'un adresi. `wr` – kontrol portundaki !WR sinyali bit numarası `rd` – kontrol portundaki !RD sinyali bit numarası `cd` – kontrol portundaki C/D sinyali bit numarası `rst` – kontrol portundaki !RST sinyali bit numarası Görüntü ünitesi RAM'i: Kütüphane mevcut RAM miktarını bilemez. Kütüphane RAM'i panellere böler; bir komple panel bir grafik panelini takip eden bir text paneli şeklindedir, programcı donanımında kaç panel olduğunu kendisi bilmelidir.
Gereklilikler	Hiçbir şey

Örnek	`T6963C_init(240, 128, 8, &PORTF, &PORTD, 5, 7, 6, 4) ;`
	
	`/*`
	`  * Ekrani asagidaki ayarlar ile baslangic durumuna getir`
	`  * 240 nokta genisliginde ve 128 nokta yuksekliginde`
	`  * 8 bit karakter genisliginde`
	`  * veri yolu PORTF'de`
	`  * kontrol yolu PORTD'de`
	`  * 5 No'lu bit !WR`
	`  * 7 No'lu bit !RD`
	`  * 6 No'lu bit C/D`
	`  * 4 No'lu bit !RST`
	`*/`

T6963C_writeData

Yapısı	`void T6963C_writeData(unsigned char data);`
Tanımı	Yordam T6963C denetleyicisine veri yazar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_writeData(AddrL);`

T6963C_writeCommand

Yapısı	`void T6963C_writeCommand(unsigned char data);`
Tanımı	Yordam T6963C denetleyicisine komut yazar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_writeCommand(T6963C_CURSOR_POINTER_SET);`

T6963C_setPtr

Yapısı	`void T6963C_setPtr(unsigned int addr, unsigned char t);`
Tanımı	Bu yordam (`p`) hafıza işaretleyicisini (`c`) komutu için ayarlar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_writeCommand(T6963C_CURSOR_POINTER_SET);`

T6963C_waitReady

Yapısı	`void T6963C_waitReady(void);`
Tanımı	Bu yordam durum byte'ını okur, hazır olana kadar döngü yapar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_waitReady();`

T6963C_fill

Yapısı	`void T6963C_fill(unsigned char data, unsigned int start, unsigned int len);`
Tanımı	Bu yordam, denetleyici belleğinin `start` başlangıç adresinden başlayarak `len` uzunluğundaki bölgesini byte `data` ile doldurur.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_fill(0x33,0x00FF,0x000F);`

T6963C_dot

Yapısı	`void T6963C_dot(int x, int y, unsigned char color);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. (x0, y0) yerleşimine yani x satırının y sütununa bir nokta koyar. Renk olarak şu değerleri geçirebilirsiniz: pcolor = T6963C_[WHITE[BLACK].
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_dot(x0, y0, pcolor);`

T6963C_write_char

Yapısı	`void T6963C_dot(int x, int y, unsigned char c);`
Tanımı	Bu yordam yazı gösterim panelinde iş görür. `c` karakterini `x` satırının ve `y` kolonuna yazar. Eski nokta ile kaynaştırma seçenekleri: mode = T6963C_ROM_MODE_[OR/EXOR/AND].
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_write_char('A',22,23,AND);`

T6963C_write_text

Yapısı	`void T6963C_write_text(unsigned char *str, unsigned char x, unsigned char y, unsigned char mode);`
Tanımı	Bu yordam yazı gösterim panelinde iş görür. `str` karakter dizisini `x` satırının `y` kolonuna yazar. mode = T6963C_ROM_MODE_[OR/EXOR/AND].
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_write_text(" GLCD KUTUPHANE ORNEGI, MERHABA !", 0, 0, T6963C_ROM_MODE_XOR);`

T6963C_line

Yapısı	`void T6963C_line(int px0, int py0, int px1, int py1, unsigned char pcolor);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. `(x0, y0)`'dan `(x1, y1)`'e bir çizgi çizer. Renk seçenekleri şunlardır : `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_line(0, 0, 239, 127, T6963C_WHITE);`

T6963C_rectangle

Yapısı	`void T6963C_rectangle(int x0, int y0, int x1, int y1, unsigned char pcolor);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. `(x0, y0)`'ve `(x1, y1)` ile verilen dikdörtgen'in kenarlarını çizer. Renk seçenekleri şunlardır : `pcolor` = T6963C_[WHITE[BLACK]].
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız..
Örnek	`T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE);`

T6963C_box

Yapısı	`void T6963C_box(int x0, int y0, int x1, int y1, unsigned char pcolor);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. `(x0, y0)` ve `(x1, y1)` ile verilen dikdörtgen'i içi dolu çizer. Renk seçenekleri şunlardır : `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_box(0, 119, 239, 127, T6963C_WHITE);`

T6963C_circle

Yapısı	`void T6963C_circle(int x, int y, long r, unsigned char pcolor);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. Merkezi `(x, y)` ve yarı-çapı `(r)` olan daire çizer. Renk seçenekleri şunlardır : `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_circle(120, 64, 110, T6963C_WHITE);`

T6963C_image

Yapısı	`void T6963C_image(const char *pic);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. Grafik alanını, parametre olarak girilen işaretçinin gösterdiği şekil verisi ile doldurur. Fonksiyona sağlanan şekil verisi GLCD ekran geometrisi ile uyumlu olmalıdır. Örneğin : 240x128 büyüklüğünde bir ekran için, giriş dizisi (240/8)*128 = 3840 bayt olmalıdır.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_image(mc);`

T6963C_sprite

Yapısı	`void T6963C_sprite(unsigned char px, unsigned char py, const char *pic, unsigned char sx, unsigned char sy);`
Tanımı	Bu yordam grafik gösterim panelinde iş görür. Dikdörtgensel grafik alanını yani (px, py)-(px + sx, py + sy) alanını işaretçinin gösterdiği şekil verisi ile doldurur. Sx ve sy değişkenleri ilgili şeklin büyüklüğü olmalıdır. İşaretçinin gösterdiği dizi sx*sy bayt olmalıdır.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_sprite(76, 4, einstein, 88, 119); // Bir sekilcik ciz`

T6963C_set_cursor

Yapısı	`void T6963C_set_cursor(unsigned char x, unsigned char y);`
Tanımı	Bu yordam imleçi `x` satırının `y` kolonuna getirir.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_set_cursor(cposx, cposy);`

T6963C_clearBit

Yapısı	`void T6963C_clearBit(char b);`
Tanımı	Kontrol bit'ini sıfırlar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_clearBit(b);`

T6963C_setBit

Yapısı	`void T6963C_setBit(char b);`
Tanımı	Kontrol bit'ini bir yapar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_setBit(b);`

T6963C_negBit

Yapısı	`void T6963C_negBit(char b);`
Tanımı	Kontrol bit'ini olumsuzlar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_negBit(b);`

T6963C_displayGrPanel

Yapısı	`void T6963C_displayGrPanel(unsigned int n);`
Tanımı	Numarası `n` olan grafik panelini gösterir.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_displayGrPanel(n);`

T6963C_displayTxtPanel

Yapısı	`void T6963C_displayTxtPanel(unsigned int n);`
Tanımı	Numarası `n` olan metin panelini gösterir.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_displayTxtPanel(n);`

T6963C_setGrPanel

Yapısı	`void T6963C_setGrPanel(unsigned int n);`
Tanımı	Panel numarası `(n)` için grafik başlangıç adresini hesaplar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_setGrPanel(n);`

T6963C_setTxtPanel

Yapısı	`void T6963C_setTxtPanel(unsigned int n);`
Tanımı	Panel numarası `(n)` için metin başlangıç adresini hesaplar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_setTxtPanel(n);`

T6963C_panelFill

Yapısı	`void T6963C_panelFill(unsigned int v);`
Tanımı	Tüm #n panelini `v` bit resmi ile doldurur (temizlemek için 0)
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_panelFill(v);`

T6963C_grFill

Yapısı	`void T6963C_grFill(unsigned int v);`
Tanımı	Grafik #n panelini `v` bit resmi ile doldurur (temizlemek için 0)
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_grFill(v);`

T6963C_txtFill

Yapısı	`void T6963C_txtFill(unsigned int v);`
Tanımı	Metin #n panelini `v + 32` karakteri ile doldurur (temizlemek için 0)
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_txtFill(v);`

T6963C_cursor_height

Yapısı	`void T6963C_cursor_height(unsigned int n);`
Tanımı	İmleç boyutunu ayarlar.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız..
Örnek	`T6963C_cursor_height(n);`

T6963C_graphics

Yapısı	`void T6963C_graphics(unsigned int n);`
Tanımı	Grafikleri açma/kapatma.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_graphics(1);`

T6963C_text

Yapısı	`void T6963C_text(unsigned int n);`
Tanımı	Metinleri açma/kapatma.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_text(1);`

T6963C_cursor

Yapısı	`void T6963C_cursor(unsigned int n);`
Tanımı	İmleci açma/kapatma.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_cursor(1);`

T6963C_cursor_blink

Yapısı	`void T6963C_cursor_blink(unsigned int n);`
Tanımı	İmlecin yanıp-sönme (blinking) özelliğini açma/kapatma.
Gereklilikler	Portlar başlangıç durumuna getirilmiş olmalıdır. `T6963C_init`'e bakınız.
Örnek	`T6963C_cursor_blink(0);`

T6963C_Init_240x128

Yapısı	`procedure T6963C_Init_240x128;`
Tanımı	T6963 tabanlı GLCD'yi (240x128 pikselli) varsayılan ayarlarla başlangıç durumuna getirir.
Örnek	`T6963C_Init_240x128;`

T6963C_Init_240x64

Yapısı	`procedure T6963C_Init_240x64;`
Tanımı	T6963 tabanlı GLCD'yi (240x64 pikselli) varsayılan ayarlarla başlangıç durumuna getirir.
Örnek	`T6963C_Init_240x64;`

Kütüphane Örneği

Aşağıdaki örnekte T6963C GLCD kütüphanesinin en gelişmiş yordamlarını kullanan demo bir program verilmiştir.

```
#include         "T6963C.h"
/*
 * bitmap resimleri ROM da sakli
 */
extern const char mc[] ;
extern const char einstein[] ;
/*
 * Kontrast guc kaynagi icin PWM sinyali doluluk oranı (duty
 * cycle)
 */
unsigned char    PWM_duty = 200 ;

void main(void)
        {
        unsigned char    panela ;       // Kullanilan panel
        unsigned int     i ;            // Genel amacli yazmac
        unsigned char    curs ;         // Imlec gorunulurlugu
        unsigned int     cposx, cposy ; // imlec x,y konumu

        TRISC = 0 ;              // port C sadece cikis
        PORTC = 0b00000000 ;     // yonga etkin, tersleme acik,
                                 // 8x8 font

//devam ediyor...
```

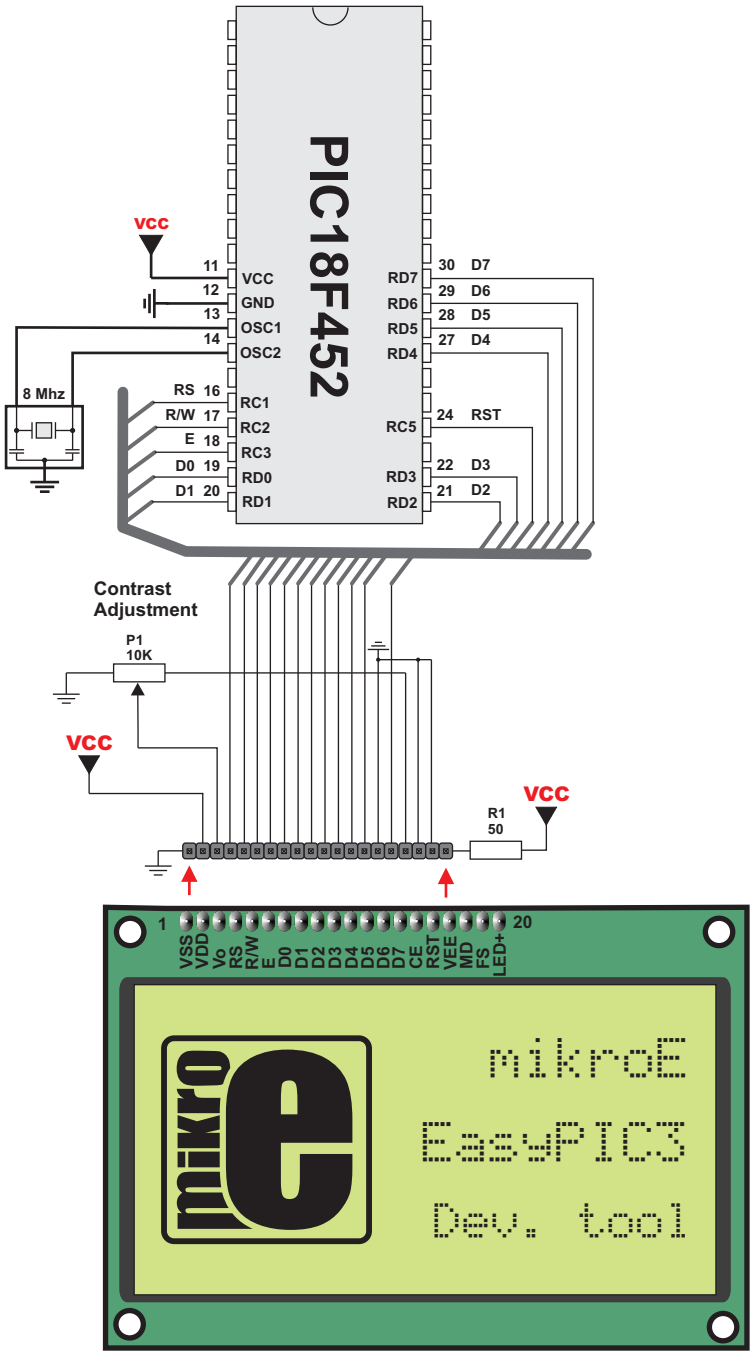
```
//devam...

      /*
         * Goruntu birimini 240 nokta genisliginde ve 128 nokta
         * yuksekliginde baslangic durumuna ayarla
         * 8 bit karakter genisligi
         * veri hatti PORTD'de
         * kontrol hatti PORTC'de
         * 3 No'lu bit !WR
         * 2 No'lu bit !RD
         * 1 No'lu bit C/D
         * 5 No'lu bit RST
      */
        T6963C_Init_240x128();
        //T6963C_init(240, 128, 8, &PORTD, &PORTC, 3, 2, 1, 5) ;
        /*
         * Ayni anda hem grafik hem de metin ekranlari etkin
         */
        T6963C_graphics(1) ;
        T6963C_text(1) ;

        panel = 0 ;
        i = 0 ;
        curs = 0 ;
        cposx = cposy = 0 ;

        /*
         * Metin mesajlari
         */
T6963C_write_text(" GLCD LIBRARY DEMO, WELCOME !", 0, 0,
T6963C_ROM_MODE_XOR) ;
        T6963C_write_text(" EINSTEIN WOULD HAVE LIKED mC", 0, 15,
T6963C_ROM_MODE_XOR) ;

        /*
         * Imlec
         */
        T6963C_cursor_height(8) ;      // 8 nokta yukseklik
        T6963C_set_cursor(0, 0) ;      // Imlec sol ust koseye
        T6963C_cursor(0) ;             // Imleci kapat
        /*
         * Dikdortgenler ciz
         */
        T6963C_rectangle(0, 0, 239, 127, T6963C_WHITE) ;
        T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE) ;
        T6963C_rectangle(40, 40, 199, 87, T6963C_WHITE) ;
        T6963C_rectangle(60, 60, 179, 67, T6963C_WHITE) ;

//devam ediyor...
```

```
//devam...
        /*
         * Bir arti isareti ciz
         */
        T6963C_line(0, 0, 239, 127, T6963C_WHITE) ;
        T6963C_line(0, 127, 239, 0, T6963C_WHITE) ;

        /*
         * Dolu kutular ciz
         */
        T6963C_box(0, 0, 239, 8, T6963C_WHITE) ;
        T6963C_box(0, 119, 239, 127, T6963C_WHITE) ;

        /*
         * Daireler ciz
         */
        T6963C_circle(120, 64, 10, T6963C_WHITE) ;
        T6963C_circle(120, 64, 30, T6963C_WHITE) ;
        T6963C_circle(120, 64, 50, T6963C_WHITE) ;
        T6963C_circle(120, 64, 70, T6963C_WHITE) ;
        T6963C_circle(120, 64, 90, T6963C_WHITE) ;
        T6963C_circle(120, 64, 110, T6963C_WHITE) ;
        T6963C_circle(120, 64, 130, T6963C_WHITE) ;

        T6963C_sprite(76, 4, einstein, 88, 119) ; // Sekilcik ciz

        T6963C_setGrPanel(1) ; // Diger grafik panelini sec

        T6963C_image(mc) ;     // grfk. panelini bir resimle doldur

        for(;;)
                {

                /*
                 * Eger RB1 butonuna basilirsa
                 * panel 0 ile panel 1 arasinda gecis yap
                 */
                if(PORTB & 0b00000010)
                        {
                        panel++ ;
                        panel &= 1 ;
                        T6963C_displayGrPanel(panel) ;
                        Delay_ms(300) ;
                        }
                                                }

//devam ediyor...
```

```
//devam...
                        /*
                        * Eger RB2 butonuna basilirsa sadece
                        * grafik panelini goruntule
                        */
                        else if(PORTB & 0b00000100)
                                {
                                T6963C_graphics(1) ;
                                T6963C_text(0) ;
                                Delay_ms(300) ;
                                }

                        /*
                        * Eger RC2 butonuna basilirsa sadece
                        * text panelini goruntule
                        */
                        else if(PORTB & 0b00001000)
                                {
                                T6963C_graphics(0) ;
                                T6963C_text(1) ;
                                Delay_ms(300) ;
                                }

                        /*
                        * Eger RB4 butonuna basilirsa
                        * text ve grafik panellerini goruntule
                        */
                        else if(PORTB & 0b00010000)
                                {
                                T6963C_graphics(1) ;
                                T6963C_text(1) ;
                                Delay_ms(300) ;
                                }

                        /*
                        * Eger RB5 butonuna basilirsa imleci degistir
                        */
                        else if(PORTB & 0b00100000)
                                {
                                curs++ ;
                                if(curs == 3) curs = 0 ;
                                switch(curs)



//devam ediyor...
```

```
// devam...

                    /*
                     * Imleci otele (gorunur degilse bile)
                     */
                    cposx++ ;
                    if(cposx == T6963C_txtCols)
                            {
                            cposx = 0 ;
                            cposy++ ;
                if(cposy == T6963C_grHeight / T6963C_CHARACTER_HEIGHT)
                                    {
                                    cposy = 0 ;
                                    }
                            }
                    T6963C_set_cursor(cposx, cposy) ;

                    Delay_ms(100) ;
                    }
            }
```

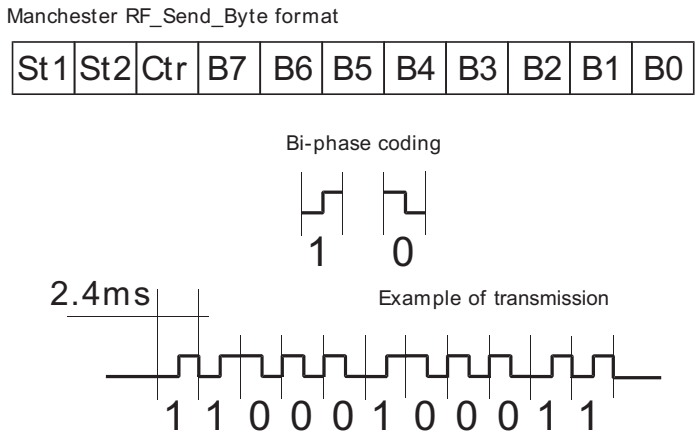
Donanım Bağlantısı



Toshiba T6963C Graphic LCD (240x128)

Manchester Kod Kütüphanesi

mikroC, Manchester kodlanmış sinyalleri iletmek için bir kütüphane sağlamaktadır. Manchester kodlaması, veri ve saat sinyallerinin bir bileşimleri yapılarak tek bir kendinden eşzamanlı veri akış yolu sağlamak için oluşturulmuş bir koddur. Her kodlanmış bit'in periyodunun orta noktası geçiş içerir. Geçişin yönü, bitin 0 veya 1 olmasını belirler. Fazın ikinci yarısı ilgili bitin gerçek değeridir, fazın ilk yarısı ise ilgili bitin tümleyenidir (evriğidir). (aşağıdaki resimde gösterildiği gibi).



Not: Manchester'in alıcı yordamları tıkanan (blocking) çağrılardır: (`Man_Receive_Config, Man_Receive_Init, Man_Receive`). Bu şu demektir, görev (task) yerine getirilene kadar (byte alınır, senkronizasyon oluşturulur v.b.) PIC bekleyecektir.

Alma yordamları 340 ~ 560 bps' lik (baud oranı olan) bir bant ile sınırlıdır.

Kütüphane Yordamları

```
Man_Receive_Config
Man_Receive_Init
Man_Receive
Man_Send_Config
Man_Send_Init
Man_Send
```

Man_Receive_Config

Yapısı	`void Man_Receive_Config(char *port, char rxpin);`
Tanımı	Fonksiyon sinyal alımı için PIC'i hazırlar. Giriş sinyalinin portunu (`port`) ve pinini (`rxpin` (0-7)) belirtmeniz gereklidir.. Alımda çok hata oluşması durumunda senkronizasyonu sağlamak için `Man_Receive_Init`'i bir kez daha çağırmalısınız.
Örnek	`Man_Receive_Config(&PORTD, 6);`

Man_Receive_Init

Yapısı	`void Man_Receive_Init(char *port);`
Tanımı	Fonksiyon sinyal alımı için PIC'i hazırlar. Sadece portu (`port`) belirtmeniz gereklidir. `rx` pin'i varsayılan olarak 6'dır. Alımda çok hata oluşması durumunda senkronizasyonu sağlamak için `Man_Receive_Init`'i bir kez daha çağırmalısınız.
Örnek	`Man_Receive_Init(&PORTD);`

Man_Receive

Yapısı	`void Man_Receive(char *error);`
Dönüş	Sinyalden bir bayt döndürür.
Tanımı	Fonksiyon sinyalden bir bayt çözer. Eğer sinyal formatı beklenenle eşleşmezse, hata bayrağı (`error`) 255'e çekilir.
Gereklilikler	Bu fonksiyonu kullanmak için, ilk önce PIC'i alım için hazırlamalısınız. `Man_Receive_Config` veya `Man_Receive_Init`'e bakınız.
Örnek	`temp = Man_Receive(error);` `if (error) { ... /* Hata degerlendirme kismi */ }`

Man_Send_Config

Yapısı	`void Man_Send_Config(char *port, char txpin);`
Tanımı	Fonksiyon sinyal yollamak için PIC'i hazırlar. Giden sinyalin portunu (`port`) ve pinini (`txpin` (0-7)) belirtmeniz gereklidir. Sinyal gönderim hızı (baud rate) sabit olup 500 bps (bit bölü saniye)'dir.
Örnek	`Man_Send_Config(&PORTD, 0);`

Man_Send_Init

Yapısı	`void Man_Send_Init(char *port);`
Tanımı	Fonksiyon sinyal yollamak için PIC'i hazırlar. Giden sinyalin sadece portunu (`port`) belirtmeniz gereklidir. `txpin başlangıçta` 0'dır. Sinyal gönderim hızı (baud rate) sabit olup 500 bps (bit bölü saniye)'dir.
Örnek	`Man_Send_Init(&PORTD);`

Man_Send

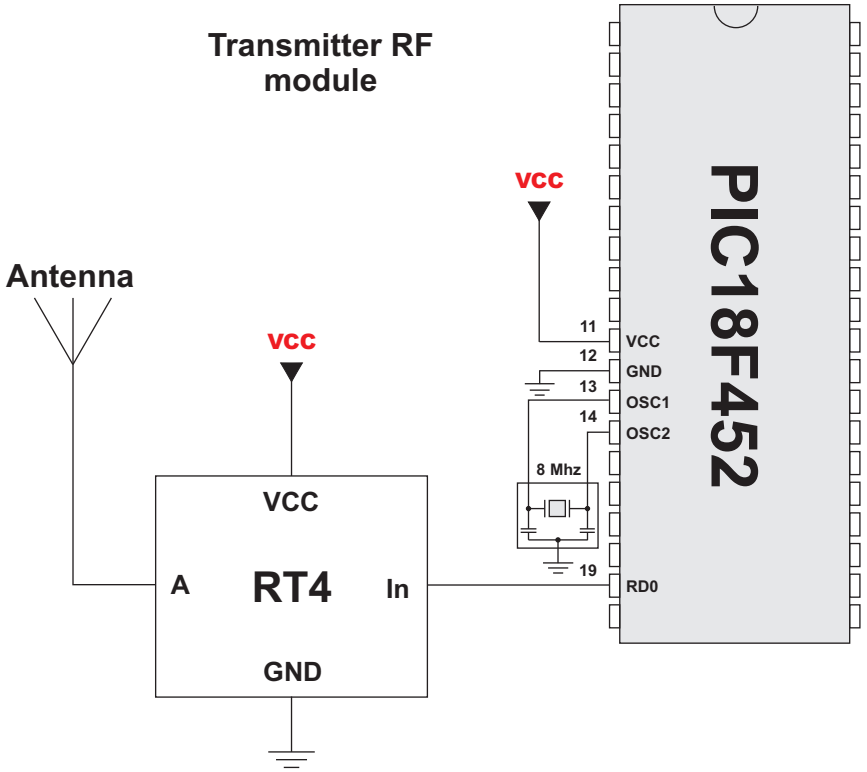
Yapısı	`void Man_Send(unsigned short data);`
Tanımı	Fonksiyon bir bayt veri (`data`) yollar.
Gereklilikler	Bu fonksiyonu kullanmak için ilk başta PIC'i gönderim için hazırlamalısınız. `Man_Send_Config` veya `Man_Send_Init`'e bakınız.

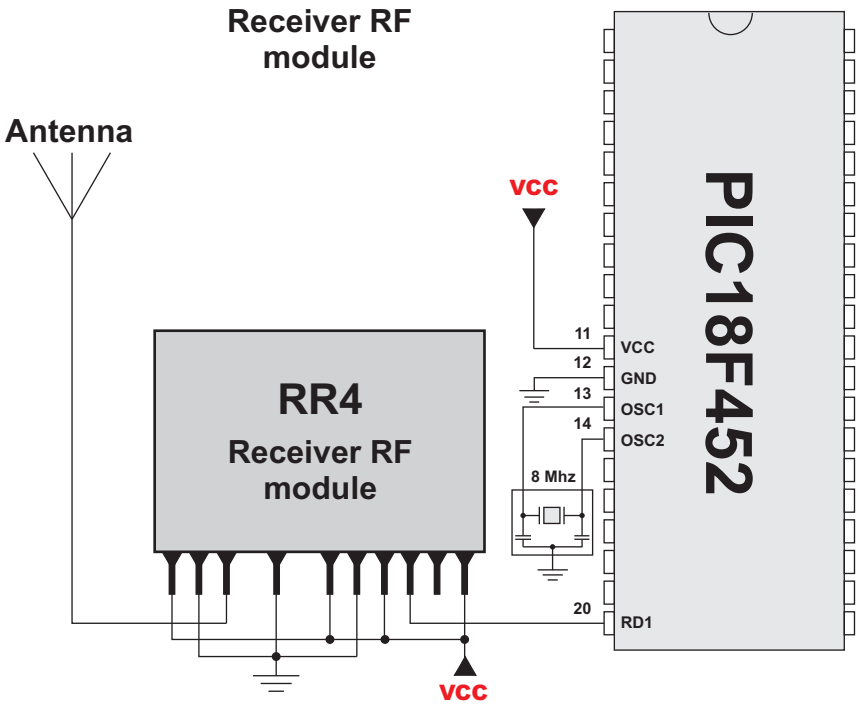
Örnek	`unsigned short msg;`
	`...`
	`Man_Send(msg);`

Kütüphane Örneği

```
unsigned short error, ErrorCount, IdleCount, temp, LetterCount;

void main() {
  ErrorCount = 0;
  TRISC      = 0;                      // Hata gostergeci
  PORTC      = 0;
  Man_Receive_Config(&PORTD, 6);       // Aliciyi senkronize et
  Lcd_Init(&PORTB);                    // LCD'yi PORTB'ye ayarla

  while (1) {                          // Sonsuz dongu (Asagidakileri daima yap)
    IdleCount = 0;                     // idle sayacini sifirla
    do {
      temp = Man_Receive(error);       // Bayt alimini dene
      if (error)
        ErrorCount++
      else
        PORTC = 0;
      if (ErrorCount > 20) {           // Eger cok fazla hata varsa
        ErrorCount = 0;                //   aliciyi tekrar senkronize et
        PORTC = 0xAA;                  // Hatayi bildir
        Man_Receive_Init(&PORTD);      // Aliciyi senkronize et
      }
      IdleCount++;
      if (IdleCount > 18) {            // Eger belli bir sure hic veri alinmazsa
        IdleCount = 0;                 //  tekrar senkronizasyonu dene
        Man_Receive_Init(&PORTD);      // Aliciyi senkronize et
        }
    } while (temp != 0x0B);            // Mesaj sonu isareti
      if (error != 255) {              // Eger hic hata yoksa mesaji yazdir
        Lcd_Cmd(LCD_CLEAR);
        LetterCount = 0;
        while (LetterCount < 17) {     // Mesaj 16 karakter uzunlugundadir
          LetterCount++;
          temp = Man_Receive(error);
          if (error != 255)
            Lcd_Chr_Cp(temp)
          else {
            ErrorCount++; break;
          }
        }
        temp = Man_Receive(error);
        if (temp != 0x0E)
          ErrorCount++;
      } // end if
    } // end while
} //~!
```

Donanım Bağlantısı




Receiver RF module
Antenna
vcc
PIC18F452
11
VCC
12
GND
13
OSC1
14
OSC2
8 Mhz
RR4
Receiver RF module
20
RD1
vcc

Multi Media Kart Kütüphanesi

mikroC, SPI haberleşme yoluyla Multi Media Kart üzerine veri erişimi için bir kütüphane sağlamaktadır. Bu kütüphane aynı zamanda Secure Digital (SD) flash bellek kartı standardını da desteklemektedir.

Not:

-Kütüphane yalnızca PIC18 ailesi ile çalışır;

-Kütüphane fonksiyonları, dosyaları yalnızca ana klasörde oluşturur veya okur;

-Kütüphane fonksiyonları dosyalara yazarken hem FAT1 hem de FAT2 tablolarını oluştururlar. Ancak dosya verisi sadece FAT1 tablosundan okunabilir. O nedenle eğer FAT1 tablosu bir sebeple bozulmuş ise tekrar kazanımı mümkün olmaz.

- Versiyon 5.0.0.3'dan itibaren, kütüphane 0. bölmede Master Boot Record (MBR)'a sahip ortamları da (media) destekler duruma gelmiştir. Gerekli olan bilgiyi okur ve daha sonra ilk uygun birincil mantıksal bölmeye (primary logical partition) atlar. MBR, fiziksel ve mantıksal sürücüler, birincil/ikincil bölmeler ve bölme tabloları hakkında daha fazla bilgi edinmek için, Wikipedia veya benzeri diğer kaynaklara bakabilirsiniz.

Not: `Mmc_Init` fonksiyonu çağırılmadan önce;
`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)` fonksiyonu çağırılmalıdır.

Kütüphane Yordamları

```
Mmc_Init
Mmc_Read_Sector
Mmc_Write_Sector
Mmc_Read_Cid
Mmc_Read_Csd

Mmc_Fat_Init
Mmc_Fat_Assign
Mmc_Fat_Reset
Mmc_Fat_Rewrite
Mmc_Fat_Append
Mmc_Fat_Read
Mmc_Fat_Write
Mmc_Set_File_Date
Mmc_Fat_Delete
Mmc_Fat_Get_File_Date
Mmc_Fat_Get_File_Size
Mmc_Fat_Get_Swap_File
```

Mmc_Init

Yapısı	`unsigned short Mmc_Init(char *port, char pin);`
Dönüş	Eğer okuma sorunsuz gerçekleşirse 0 döndürür. Eğer hata oluşmuşsa 1 döndürür.
Tanımı	Yonga seçme pini `port` ve `pin` parametreleri ile verilen MMC'yi başlangıç durumuna getirir. İlgili MCU için haberleşme port ve pinleri SPI donanım ayarları ile verilir. Eğer MMC kart takılı ve sorunsuz bir şekilde başlangıç durumuna getirilebilmişse 0 döndürür. Aksi halde 1 döndürür. `Mmc_Init`, bu kütüphanenin diğer fonksiyonları kullanılmadan önce çağırılmalıdır.
Gereklilikler	`Mmc_Init` yordamından önce `Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)` çağırılmalıdır.
Örnek	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);` `while (Mmc_Init(&PORTC,2));  // MMC ayarlanana kadar dongu yap`

Mmc_Read_Sector

Yapısı	`unsigned short Mmc_Read_Sector(unsigned long sector, char *data);`
Dönüş	Eğer okuma sorunsuz gerçekleşirse 0 döndürür. Eğer hata oluşmuşsa 1 döndürür.
Tanımı	Fonksiyon MMC kartın `sector` ile verilen adresinden bir bölme (sektör yani 512 bayt) okur. Okunan veri `data` dizisine depolanır. Fonksiyon eğer okuma sorunsuz gerçekleşirse 0 döndürür. Eğer hata oluşmuşsa 1 döndürür.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Init`'e bakınız.
Örnek	`error = Mmc_Read_Sector(sector, data);`

Mmc_Write_Sector

Yapısı	`unsigned short Mmc_Write_Sector(unsigned long sector,char *data);`
Dönüş	Eğer yazma sorunsuz gerçekleşmiş ise 0 döndürür. Eğer yazma komutunun gönderiminde hata varsa 1 döndürür. Eğer yazmada bir sorun varsa 2 döndürür.
Tanımı	Fonksiyon MMC kartın `sector` ile verilen adresine bir bölme (512 byte) veri `data` yazar. Fonksiyon eğer yazım sorunsuz gerçekleşmiş ise 0 döndürür. Eğer yazma komutunun gönderiminde hata varsa 1 döndürür. Eğer yazımda bir sorun varsa 2 döndürür.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Init`'e bakınız.
Örnek	`error = Mmc_Write_Sector(sector, data);`

Mmc_Read_Cid

Yapısı	`unsigned short Mmc_Read_Cid(unsigned short *data_for_registers);`
Dönüş	Eğer okuma sorunsuz gerçekleşirse 0 döndürür. Eğer hata oluşmuşsa 1döndürür.
Tanımı	Fonksiyon, CID yazmacını okur ve yazmacın 16 baytlık içeriğini `data_for_registers`'a döndürür.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Init`'e bakınız.
Örnek	`error = Mmc_Read_Cid(data);`

Mmc_Read_Csd

Yapısı	`unsigned short Mmc_Read_Csd(unsigned short *data_for_registers);`
Dönüş	Eğer okuma sorunsuz gerçekleşirse 0 döndürür. Hata oluşmuşsa 1döndürür.
Tanımı	Fonksiyon, CSD yazmacını okur ve yazmacın 16 baytlık içeriğini `data_for_registers`'a döndürür.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Init`'e bakınız.
Örnek	`error = Mmc_Read_Csd(data);`

Mmc_Fat_Init

Yapısı	`unsigned short Mmc_Fat_Init(unsigned short *port, unsigned short pin);`
Dönüş	Eğer sorunsuz bir şekilde başlatılmışsa 0 döndürür. Eğer kök bölme (sektör) bulunamazsa 1, kart algılanamazsa 255 döndürür.
Tanımı	FAT yordamları için MMC/SD kartlarını başlangıç durumuna getirir; haberleşme için CS hattı port ve pin parametreleri sayesinde verilir. Bu fonksiyon MMC FAT kütüphanesinin diğer fonksiyonları kullanılmadan önce çağırılmalıdır.
Gereklilikler	`Mmc_Fat_Init` yordamından önce; `Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)` çağırılmalıdır.

Örnek	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);`
	`// RC2'de MMC FAT baslangic durumuna gelinceye kadar dongu yap.`
	`while (Mmc_Fat_Init(&PORTC, 2)) ;`

Mmc_Fat_Assign

Yapısı	`void Mmc_Fat_Assign(char *filename);`
Tanımı	Fonksiyon bizim çalışacağımız dosyaya ulaşır (belirler). Fonksiyon `filename` ile belirtilen dosyayı kök dizinde arar. Eğer dosya bulunursa, yordam dosyanın başlama bölmesi ve boyutu gibi parametreleri edinerek başlamaya hazır duruma getirir. Eğer dosya bulunmazsa, izin verildiği taktirde, verilen isimle yeni bir boş dosya, oluşturulacaktır. Dosya ismi (`filename`) büyük harfle yazılmalı ve 8+3 karakterli olmalıdır. 8 harflik dosya adı ile 3 harflik dosya uzantısı arasında nokta kullanılmadığına dikkat ediniz.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Fat_Init`'e bakınız.

Örnek	`// MMC'nin kok dizininde "EXAMPLE1.TXT" dosyasina ulas.`
	`// Eger dosya bulunamazsa, yordam yeni bir tane olusturur.`
	`Mmc_Fat_Assign("EXAMPLE1TXT");`

Mmc_Fat_Reset

Yapısı	`void Mmc_Fat_Reset(unsigned long *size);`
Tanımı	Fonksiyon atanmış (ulaşılmış) olan dosyanın dosya işaretçisini resetler (dosyanın başlangıcına taşır), böylece dosya okunabilir. `size` parametresi boyutu atanmış (ulaşılmış) dosyanın boyutunu bayt olarak depolar.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Fat_Init`'e bakınız.
Örnek	`Mmc_Fat_Reset(&filesize);`

Mmc_Fat_Rewrite

Yapısı	`void Mmc_Fat_Rewrite(void);`
Tanımı	Fonksiyon dosya işaretçisini resetler ve atanmış (ulaşılmış) dosyayı temizler. Böylece dosyaya yeni veri yazılabilir.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Fat_Init`'e bakınız.
Örnek	`Mmc_Fat_Rewrite();`

Mmc_Fat_Append

Yapısı	`void Mmc_Fat_Append(void);`
Tanımı	Yordam dosya işaretçisini belirtilen dosyanın sonuna taşır. Böylece dosyaya veri eklenebilir.
Gereklilikler	Kütüphane başlangıç durumuna getirilmiş olmalıdır. `Mmc_Fat_Init`'e bakınız.
Örnek	`Mmc_Fat_Append();`

Mmc_Fat_Read

Yapısı	`void Mmc_Fat_Read(unsigned short *data);`
Tanımı	Yordam, dosya işaretçisinin gösterdiği baytı okur ve bunu `data`'ya depolar. Dosya işaretçisi her `Mmc_Fat_Read` çağırıldığında otomatik olarak 1 artacaktır.
Gereklilikler	Dosya işaretçisi başlangıç durumuna getirilmiş olmalıdır; `Mmc_Fat_Reset`'e bakınız.
Örnek	`Mmc_Fat_Read(&mydata);`

Mmc_Fat_Write

Yapısı	`void Mmc_Fat_Write(char *fdata, unsigned data_len);`
Tanımı	Yordam atanmış (ulaşılmış) dosyaya bayt öbeğini (`fdata`) dosya işaretçisinin gösterdiği yerde yazar.
Gereklilikler	Dosya işaretçisi başlangıç durumuna getirilmiş olmalıdır; `Mmc_Fat_Append` veya `Mmc_Fat_Rewrite`' a bakınız.
Örnek	`Mmc_Fat_Write(txt, 21);` `Mmc_Fat_Write("Hello\nworld", 1);`

Mmc_Set_File_Date

Yapısı	`void Mmc_Set_File_Date(unsigned year, char month, char day, char hours, char min, char sec);`
Tanımı	Sistem zaman belirtecini (timestamp)'ını bir dosyaya yazar. Bu yordamı bir dosyaya yazmadan önce muhakkak kullanın. Aksi taktirde dosyaya bilinmeyen bir zaman belirteci eklenecektir.
Gereklilikler	Dosya işaretçisi başlangıç durumuna getirilmiş olmalıdır; `Mmc_Fat_Append` veya `Mmc_Fat_Rewrite`' a bakınız.
Örnek	`// 1 Nisan 2005, 18:07:00` `Mmc_Set_File_Date(2005, 4, 1, 18, 7, 0);`

Mmc_Fat_Delete

Yapısı	`void Mmc_Fat_Delete();`
Tanımı	MMC'den dosyaları siler.
Gereklilikler	MMC ile FAT işlemleri için portlar başlangıç durumuna getirilmiş olmalıdırlar. `Mmc_Fat_Init`'e bakınız. Dosya atanmış (yani dosyaya ulaşılmış) olmalıdır. `Mmc_Fat_Assign`'a bakınız.
Örnek	`Mmc_Fat_Delete();`

Mmc_Fat_Get_File_Date

Yapısı	`void Mmc_fat_Get_File_Date(unsigned int *year, unsigned short *month, unsigned short *day, unsigned short *hours, unsigned short *mins);`
Tanımı	Dosyanın zaman özelliklerini okur. Dosyanın yılını, ayını, gününü, saatini ve dakikasını okuyabilirsiniz.
Gereklilikler	MMC ile FAT işlemleri için portlar başlangıç durumuna getirilmiş olmalıdırlar. `Mmc_Fat_Init`'e bakınız. Dosya atanmış (yani dosyaya ulaşılmış) olmalıdır. `Mmc_Fat_Assign`'a bakınız.
Örnek	`Mmc_Fat_Get_File_Date(year, month, day, hours, mins);`

Mmc_Fat_Get_File_Size

Yapısı	`unsigned long Mmc_fat_Get_File_Size();`
Tanımı	Aktif dosyanın boyutunu (byte olarak) döndürür.
Gereklilikler	MMC ile FAT işlemleri için portlar başlangıç durumuna getirilmiş olmalıdırlar. `Mmc_Fat_Init`'e bakınız. Dosya atanmış (yani dosyaya ulaşılmış) olmalıdır. `Mmc_Fat_Assign`'a bakınız.
Örnek	`Mmc_Fat_Get_File_Size();`

Mmc_Fat_Get_Swap_File

Yapısı	`unsigned long Mmc_Fat_Get_Swap_File(unsigned long sectors_cnt);`
Dönüş	Eğer değiş-tokuş (swap) dosyası oluşturulmuşsa yeni oluşturulmuş değiş-tokuş dosyasının başlangıç bölmesi numarasını döndürür. Aksi halde fonksiyon sıfır döndürür.
Tanımı	Bu fonksiyon MMC veya SD bellekler üzerinde yeni bir değiş-tokuş dosyası (swap file) oluşturmak için kullanılır. Bu fonksiyon kullanıcının değiş-tokuş dosyasında olmasını istediği peşpeşe bölmelerin sayısını `sector_cnt` argumanı olarak kabul eder. İşlem boyunca, fonksiyon peşpeşe bölmelerin varlığını araştırır. Bunların sayısı `sector_cnt` tarafından belirtilir. Eğer ortamda böyle uygun bir alan varsa, değiştirme dosyası MIKROSWP.SYS olarak adlandırılarak oluşturulur ve bu alan onun için atanır (FAT tablolarında). Bu dosyanın öznitelikleri şunlardır: sistem dosyası, arşiv dosyası ve gizli dosya. Eğer MIKROSWP.SYS dosyası bu ad ile zaten ortamda mevcut ise, bu fonksiyon yenisini oluşturunca eskisini siler. Değiş-tokuş dosyalarının amacı `Mmc_Read_Sector` ve `Mmc_Write_Sector` fonksiyonlarını doğrudan kullanarak ve FAT sistemini bozmayarak, MMC/SD belleklerine olağandan daha hızlı bir şekilde okuma ve yazma işlemini gerçekleştirmektir. Değiş-tokuş dosyaları kullanıcının veriyi üzerinde kolayca okuyup yazabildiği bir pencere(window) gibi düşünülebilir. mikroC kütüphanelerinin ana amacı hızlı veri toplama işlemini gerçekleştirmektir; zamanlamanın önemli olduğu veri toplama işlemi gerçekleştikten sonra bu veri normal bir dosyaya tekrar yazılabilir ve en uygun yolla biçimlendirilebilir.
Gereklilikler	MMC ile FAT işlemleri için portlar başlatılmalıdırlar. `Mmc_Fat_Init`'e bakınız.
Örnek	

```
/*
 * Bir degis-tokus dosyasi olusturmayi dener.
 * Bu dosyanin boyutu en az 1000 bolme olacaktir.
 * Eger islem basarili olursa USART uzerinden başlangıç
 * bolmesinin numarasi gonderilir.
*/
void M_Create_Swap_File() {
  size = Mmc_Fat_Get_Swap_File(1000);
  if (size) {
    Usart_Write(0xAA);
    Usart_Write(Lo(size));
    Usart_Write(Hi(size));
    Usart_Write(Higher(size));
    Usart_Write(Highest(size));
    Usart_Write(0xAA);
  }
}//~
```

Kütüphane Örneği

Aşağıdaki kod MMC kütüphane yordamlarını test eder. İlk olarak, tamponu (buffer) 512 tane "M" karakteri ile dolduracağız ve bunu 55'inci bölmeye (sector) yazacağız; daha sonra, diziye "E" karakteri ile devam edeceğiz ve 56'ıncı bölmeye yazacağız. Son olarak, 55'inci ve 56'ıncı bölmeleri, yazımın hatasız gerçekleşip gerçekleşmediğini kontrol etmek için, okuyacağız.

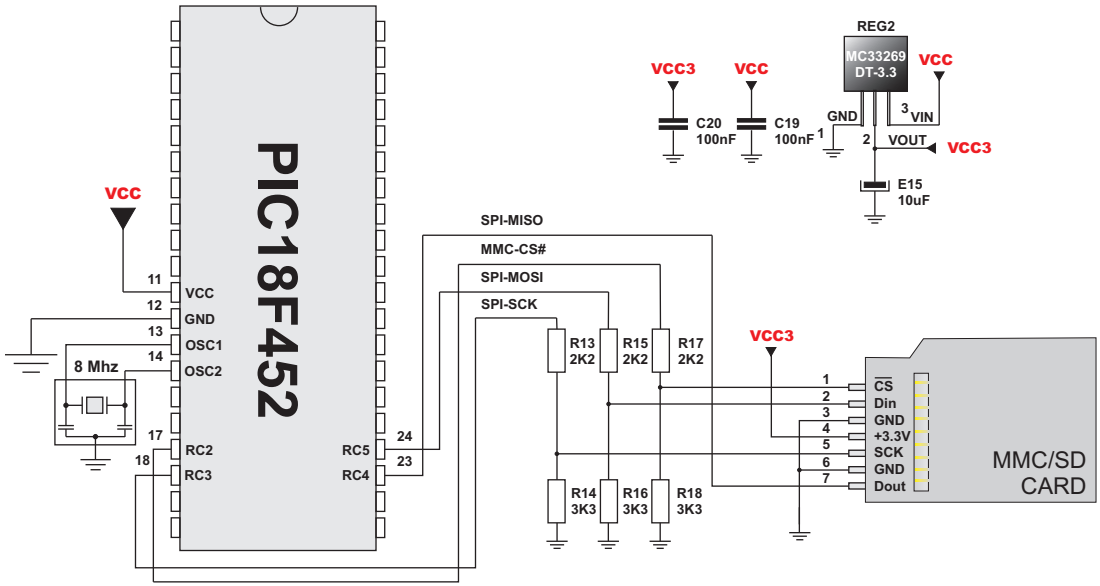
```
unsigned i;
unsigned short tmp;
unsigned short data[ 512];

void main() {
  Usart_Init(9600);

Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW,LOW_2_HIGH);
  // SPI'yi baslat

  // MMC baslangic durumuna getirilene kadar bekle
  while (Mmc_Init(&PORTC, 2)) ;

  // Tamponu "M" karakteri ile doldur
  for (i = 0; i <= 511; i++) data[ i] = "M";

  // Tamponu MMC kartinin 55'inci bolmesine yaz
  tmp = Mmc_Write_Sector(55, data);

  // Tamponu "E" karakteri ile doldur
  for (i = 0; i <= 511; i++) data[ i] = "E";

  // Tamponu MMC kartinin 56'inci bolmesine yaz
  tmp = Mmc_Write_Sector(56, data);

  // 55'inci bolmeden oku
  tmp = Mmc_Read_Sector(55, data);

  // 512 bayti tampondan USART'a gonder
  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[ i]);

  // 56'inci bolmeden oku
  tmp = Mmc_Read_Sector(56, data);

  // 512 bayti tampondan USART'a gonder

  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[ i]);

} //~!
```

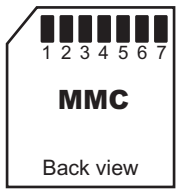
Kütüphane Örneği

Bir sonraki program MMC FAT yordamlarını test eder. İlk olarak, MMC kartın kök dizininde 5 farklı dosya oluşturacağız ve içeriğini değişik bilgilerle dolduracağız. Daha sonra, dosyayı okuyacağız ve USART yoluyla kontrol etmek için yollayacağız.

```
char FAT_ERROR[20] = "FAT16 bulunmadı";
char file_contents[50] = "XX MMC/SD FAT16 library by Anton Rieckert";
char filename[14] = "MIKRO00xTXT";  // dosya ismi
unsigned short tmp, character, loop;
long i, size;
void main() {
  PORTB = 0;
  TRISB = 0;
  Usart_Init(19200);  // USART'i dosyaların okunmasi icin ayarla
Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW,LOW_2_HIGH);
  // SPI'yi baslat
  if (!Mmc_Fat_Init(&PORTC, 2)) {   // FAT'i bulmayı dene
    tmp = 0;
    while (FAT_ERROR[tmp])
      Usart_Write(FAT_ERROR[tmp++]);
  }
  for (loop = 1; loop <= 5; loop++) {   // MMC uzerinde 5 dosya istiyoruz
    filename[7] = loop + 64;            // Sayilari ayarla 1, 2, 3, 4, veya 5
    Mmc_Fat_Assign(&filename,1);        // Eger dosya bulunamazsa yenisini olustur
    Mmc_Fat_Rewrite();                  // Dosyayi temizle ve yazmaya hazirla
    file_contents[0] = loop / 10 + 48;
    file_contents[1] = loop % 10 + 48;
    Mmc_Fat_Write(file_contents, 41);   // Ulasilmis dosyaya veriyi yaz
    Mmc_Fat_Append();                   // Dosyaya daha fazla veri ekleyecegiz
    Mmc_Fat_Write(file_contents, 41);   // Veriyi dosyaya yaz
    Delay_ms(200);
  }
  // Ayni dosyalara daha fazla veri eklemek istiyorsak
  for (loop = 1; loop <= 5; loop++) {
    filename[7] = loop + 64;
    Mmc_Fat_Assign(&filename, 1);       // Bir dosyaya ulas
    Mmc_Fat_Append();
    Mmc_Fat_Set_File_Date(2005,6,21,10,loop,0);
    Mmc_Fat_Write(" mikroElektronika icin 2005\r\n", 30);
    Mmc_Fat_Append();
    Mmc_Fat_Write(file_contents, 41);
    Mmc_Fat_Reset(&size);               // Dosyayi okumak icin, dosya boyutunu dondur
    for (i = 1; i <= size; i++) {       // USART'a tum dosyayi yaz
      Mmc_Fat_Read(&character);
      Usart_Write(character);
    }
    Delay_ms(200);
  }
} //~!
```

Donanım Bağlantısı





Tek-Tel (OneWire) Kütüphanesi

Tek-Tel kütüphanesi yordamları PIC MCU'nun Tek-Tel hattı yoluyla haberleşmesini sağlar; örneğin DS1820 dijital termometre ile haberleşme gibi. Bu bir Efendi (Master)/Köle(Slave) protokolüdür ve gereken tüm kablolama tek bir teldir. Kullanılan bu donanım konfigürasyonundan dolayı (yani tekli pullup bağlantısı ve açık kollektör sürücüleri), köleler kendilerine gerekli olan gücü bile bu tel üzerinden elde edebilirler.

Bu protokolün bazı temel özellikleri :

- tek efendi sistem,
- düşük maliyet,
- düşük aktarım oranı (16 kbps'a kadar),
- Oldukça uzun bağlantı uzaklığı (300 metre'ye kadar),
- küçük veri transfer paketleri.

Her Tek-Tel cihaz tek bir 64 bitlik kayıt numarasına sahiptir (8-bit cihaz tipi, 48-bit seri numarası ve 8-bit çevrimsel hata denetimi; CRC). Bu nedenle çoklu köleler aynı hat üzerinde var olabilirler.

Not: Yordamların Dallas dijital termometreleri ile kulanılabilmesi için Osilatör frekansı Fosc, en az 4 MHz olmalıdır.

Kütüphane Yordamları

```
Ow_Reset
Ow_Read
Ow_Write
```

Ow_Reset

Yapısı	`char Ow_Reset(char *port, char pin);`
Dönüş	DS1820 takılıysa 0, takılı değil ise 1 döndürür.
Tanımı	DS1820 için Tek-Tel reset sinyali yayımlar. `port` ve `pin` parametreleri DS1820'nin bağlantısını belirtir.
Gereklilikler	Sadece Dallas DS1820 sıcaklık sensörü ile çalışır.
Örnek	`Ow_Reset(&PORTA, 5); // RA5'e bagli olan DS1820'yi resetle`

Ow_Read

Yapısı	`char Ow_Read(char *port, char pin);`
Dönüş	Dışardaki cihazdan Tek-Tel hattı üzerinden okunan veri.
Tanımı	Veri'nin bir byte'ını Tek-Tel hattı üzerinden okur.
Örnek	`tmp = Ow_Read(&PORTA, 5);`

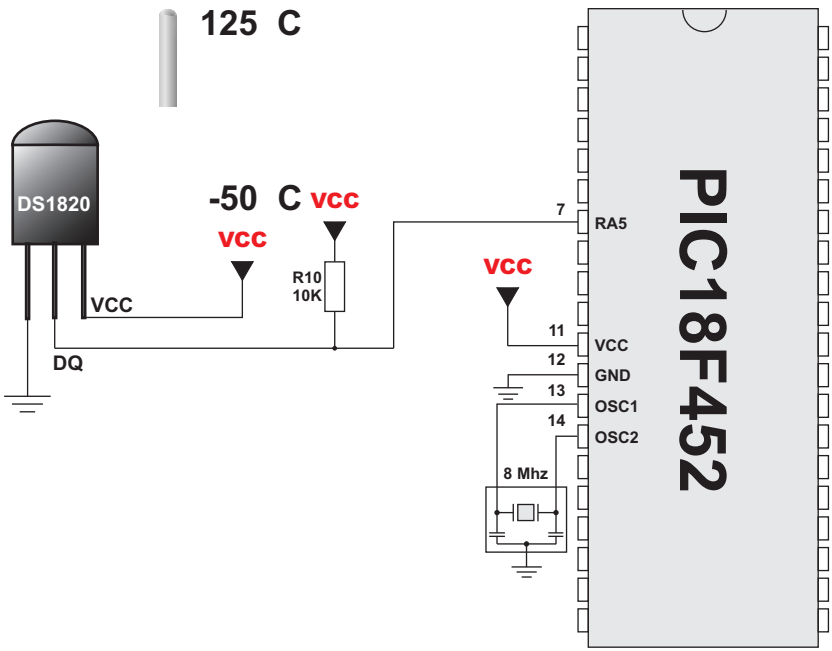
Ow_Write

Yapısı	`void Ow_Write(char *port, char pin, char par);`
Tanımı	Veri'nin bir byte'ını (`par` bağımsız değişkeni) Tek-Tel hattı üzerinden yazar.
Örnek	`Ow_Write(&PORTA, 5, 0xCC);`

Kütüphane Örneği

```
unsigned temp;
unsigned short  j;

void Display_Temperature(unsigned int temp) {
  //...
}

void main() {
  ADCON1 = 0xFF;                // RA5 pin'ini dijital Giris/Cikis olarak ayarla
  PORTA  = 0xFF;
  TRISA  = 0x0F;                // PORTA giris
  PORTB  = 0;
  TRISB  = 0;                   // PORTB cikis

  // LCD'yi PORTB'de baslangic durumuna getir ve cikis icin hazirla

  do {

    OW_Reset(&PORTA,5);         // Tek-Tel reset sinyali
    OW_Write(&PORTA,5,0xCC);    // SKIP_ROM komutu yolla
    OW_Write(&PORTA,5,0x44);    // CONVERT_T komutu yolla
    Delay_us(120);

    OW_Reset(&PORTA,5);
    OW_Write(&PORTA,5,0xCC);    // SKIP_ROM komutu yolla
    OW_Write(&PORTA,5,0xBE);    // READ_SCRATCHPAD komutu yolla
    Delay_ms(400);

    j = OW_Read(&PORTA,5);      // Sicakligin LSB'sini al
    temp = OW_Read(&PORTA,5);   // Sicakligin MSB'sini al
    temp <<= 8; temp += j;      // Sonucu olustur
    Display_Temperature(temp);  // Sonucu duzenle ve LCD'de goster
    Delay_ms(500);

  } while (1);


}//~!
```

Donanım Bağlantısı



PS/2 Kütüphanesi

mikroC yaygın olarak kullanılan PS/2 klavye ile haberleşme için bir kütüphane sağlar. Kütüphane veri okumaları için kesme (interrupt) kullanmaz ve gerekli osilatör saat frekansı 6 MHz ve üzeri olmalıdır.

Kütüphane Yordamları

```
Ps2_Init
Ps2_Config
Ps2_Key_Read
```

Ps2_Init

Yapısı	`void Ps2_Init(unsigned short *port);`
Tanımı	Portu, PS/2 klavye ile kullanmak için, "varsayılan" pin ayarları ile başlangıç durumuna getirir. Varsayılan olarak `port`' un 0'ıncı pini veri (data) hattı ve 1'inci pini saat (clock) hattıdır. PS/2 kütüphanesinin diğer yordamlarını kullanmadan önce `Ps2_Init` veya `Ps2_Config`'i çağırmanız gerekmektedir.
Gereklilikler	Veri ve saat hatlarının pull-up durumda olmaları gerekmektedir.

Ps2_Config

Yapısı	`void Ps2_Config(char *port, char clock, char data);`
Tanımı	Portu, PS/2 klavye ile kullanmak için, "özel" pin ayarları ile başlangıç durumuna getirir. `data` ve `clock` parametreleri data ve saat veri hatları için ilgili portun kullanılacak pinlerini göstermektedirler. pin numaraları olan `data` ve `clock` parametreleri 0 ile 7 arasında değer almalıdırlar ve ikisi aynı sayı olmamalıdır. PS/2 kütüphanesinin diğer yordamlarını kullanmadan önce `Ps2_Init` veya `Ps2_Config`'i çağırmanız gerekmektedir.
Gereklilikler	Veri ve saat hatlarının pull-up durumda olmaları gerekmektedir.
Örnek	`Ps2_Config(&PORTB, 2, 3);`

Ps2_Key_Read

Yapısı	`char Ps2_Key_Read(char *value, char *special, char *pressed);`
Dönüş	Eğer klavyeden bir tuşu okuma sorunsuz gerçekleşmişse 1 döndürür, aksi halde 0 döndürür.
Tanımı	Yordam tuş basım bilgisine erişir. `value` parametresi basılan tuşun değerini tutar. Karakterler, sayılar, noktalama işaretleri ve boşluk değerleri için `value` bunların ASCII karşılıklarını saklayacaktır. Yordam, Shift ve Caps Lock işlevlerini tanır ve ona göre davranır. `special` parametresi özel fonksiyon tuşları (F1, Enter, Esc gibi) için bir bayraktır (flag). Eğere basılan tuş bunlardan biri ise `special` parametresi 1'e ayarlanacaktır; aksi takdirde 0 olarak kalır. `pressed` parametresi, tuşa basıldığında 1 olur, serbest bırakıldığında 0 olur.
Gereklilikler	PS/2 tuş takımı başlangıç durumuna getirilmelidir; `Ps2_Init`'e bakınız.

Örnek	`// Devam etmek icin enter'a basiniz:`
	`do {`
	`  if (Ps2_Key_Read(&value, &special, &pressed)) {`
	`  if ((value == 13) && (special == 1)) break;`
	`  }`
	`} while (1);`

Kütüphane Örneği

Bu basit örnek PS/2 klavyesinden basılan değerleri okur ve onları USART yoluyla gönderir.

Not : EasyPIC5 geliştirme kartında PS2 bağlantıları olarak RC0 pini veri hattı(data) ve RC1 pini saat hattı (clock) olarak kullanılmıştır, bu yüzden başlangıç durumunu PORTC ile ayarlıyoruz.

```
unsigned short keydata, special, down;

void main() {
  CMCON  = 0x07;          // Analog karsilastiricilari etkisiz hale getir.
                          // (PIC18 serisi icin gerekmez)
  INTCON = 0;             // Butun kesmeleri etkisiz hale getir
  Usart_Init(19200);      // USART'i baslangic durumuna ayarla
  Ps2_Init(&PORTC);       // PS/2 klavyesini PORTC'da basl. durumuna getir
  Delay_ms(100);          // Keyboardin hazir olabilmesi icin sure tani

  do {
    if (Ps2_Key_Read(&keydata, &special, &down)) {
      if (down && (keydata == 16)) { // backspace tusu
        // ...backspace tusu alindiginda yapilacaklar ...
      }
      else if (down && (keydata == 13)) { // Enter tusu
        Usart_Write(13);
      }
      else if (down && !special && keydata) {
        Usart_Write(keydata);
      }
    }
    Delay_ms(15);         // Yanlis alimlari engellemek icin ara ver.
  } while (1);
} //~!
```

PWM Kütüphanesi

CCP birimi bazı PIC MCU'larında mevcuttur. mikroC, bu birimdeki PWM donanım özelliğinin kullanımını kolaylaştıracak bir kütüphane sağlar.

Not: CCP açılımı Capture-Compare-PWM yani Yakala-Karşılaştır-PWM
Not: PWM açılımı (Pulse Width Modulation yani Darbe Genişlik Kiplenimi)

Not: İki CCP birimi olan bazı PIC MCU'lar; örneğin PIC18F8520, sizden hangi modülü kullandığınızı belirtmenizi isteyecektir; bu durumda Pwm sözcüğüne basitce 1 veya 2 ekleyiniz. Örneğin `Pwm2_Start`. Ayrıca daha düşük versiyon derleyicilerle uyumluluk ve kod yönetimi açısından, birden fazla PWM modülü içeren PIC MCU'larda PWM kütüphanesi PWM1 ile aynıdır (Dolayısıyla CCP1'i başlatmak için `Pwm1_Init` yerine `Pwm_Init` kullanabilirsiniz).

Kütüphane Yordamları

```
Pwm_Init
Pwm_Change_Duty
Pwm_Start
Pwm_Stop
```

Pwm_Init

Yapısı	`void Pwm_Init(long freq);`
Tanımı	PWM birimini; Sinyal-Boşluk-Oranı (Duty ratio) sıfır ile başlangıç durumuna getirir. `freq` parametresi Hz cinsinden istenilen PWM frekansıdır. (uygun osilatör frekansı Fosc için PIC MCU'nun kullanma kılavuzuna bakınız). PMW kütüphanesinden başka fonksiyonların kullanılabilmesi için önce `Pwm_Init` çağırılmalıdır.
Gereklilikler	Bu kütüphaneyi kullanabilmek için bir CCP birimine ihtiyacınız olacaktır. Alternatif çözümler için mikroC kurulum dosyasına bakınız; alt klasör "Examples".
Örnek	`Pwm_Init(5000);  // PWM birimini 5KHz'de baslat`

Pwm_Change_Duty

Yapısı	`void Pwm_Change_Duty(char duty_ratio);`
Tanımı	PWM doluluk (duty) oranını değiştirir. `duty_ratio` parametresi 0'dan 255'e kadar değerler alır, burada 0 %0'ı, 127 %50'yi ve 255 %100'ü göstermektedir. Diğer özel değerler (%x * 255) / 100 şeklinde hesaplanır.
Gereklilikler	Bu kütüphaneyi kullanabilmek için port C üzerinde bir CCP birimine ihtiyacınız olacaktır. Bu fonksiyonu kullanabilmek için modül başlangıç durumuna getirilmiş olmalıdır. `Pwm_Init`'e bakınız.
Örnek	`Pwm_Change_Duty(192);  // duty oranını %75'e kur`

Pwm_Start

Yapısı	`void Pwm_Start(void);`
Tanımı	PWM'i başlatır.
Gereklilikler	Bu kütüphaneyi kullanabilmek için port C üzerinde bir CCP birimine ihtiyacınız olacaktır . Bu fonksiyonu kullanabilmek için modül başlangıç durumuna getirilmiş olmalıdır. `Pwm_Init`'e bakınız.
Örnek	`Pwm_Start();`

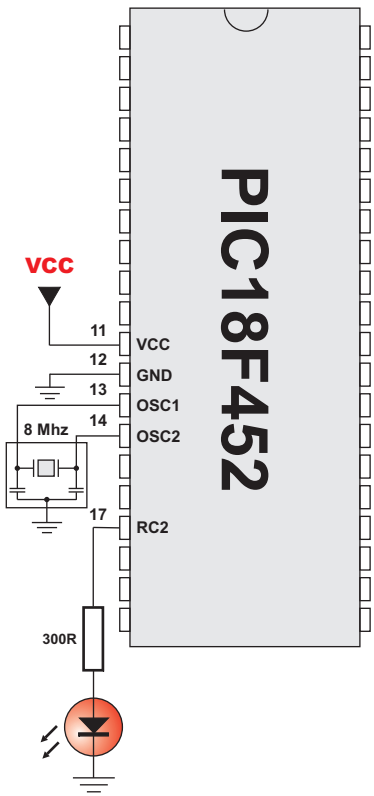
Pwm_Stop

Yapısı	`void Pwm_Stop(void);`
Tanımı	PWM'i durdurur.
Gereklilikler	Bu kütüphaneyi kullanabilmek için port C üzerinde bir CCP birimine ihtiyacınız olacaktır . Bu fonksiyonu kullanabilmek için modül başlangıç durumuna getirilmiş olmalıdır. `Pwm_Init`'e bakınız.
Örnek	`Pwm_Stop();`

Kütüphane Örneği

```
/* Asagidaki ornekte RC2 pininde PWM doluluk (duty) orani surekli degistirilir.
   Eger RC2 pinine bir LED bagli ise isigin parlakliginin degisimi
   gozlenebilir. */

unsigned char i = 0, j = 0;

void main() {
  PORTC = 0;        // PORTC cikis olarak ayarlandi
  Pwm_Init(5000);   // PWM birimini ayarlar, freq = 5kHz.
  Pwm_Start();      // PWM'i baslat

  while (1) {
    // Yavaslat, boylece LED'deki parlaklik degisimi gorunsun:
    for (i = 0; i < 20; i++) Delay_us(500);
    j++;
    Pwm_Change_Duty(j);   // Doluluk (duty) oranini degistir
  }
}
```

Donanım Bağlantısı



RS-485 Kütüphanesi

RS-485 tek bir kabloya birden fazla cihazın bağlanmasına olanak sağlayan, çok noktalı haberleşme sistemidir. mikroC Master/Slave mimarisini kullanarak RS-485 ile daha rahat çalışmayı sağlayan bir kütüphane seti sağlamaktadır. Master ve Slave cihazları bilgi paketlerini değiştirirler, her paket senkronizasyon baytlarını, CRC baytını, adres baytını ve veriyi içermektedir. Her Slave kendi adresine sahiptir ve sadece kendine yollanan paketleri alır. Slave haberleşmeyi başlatamaz. Aynı anda 485 çoklu-hat'tı yoluyla sadece bir cihazın veri yolladığından emin olmak, programlayıcının sorumluluğundadır.

RS-485 yordamları Port C üzerinde USART birimine ihtiyaç duyar. USART pinleri RS-485 arabirim alıcı-vericisine (interface transceiver) bağlanmalıdır. (LTC485 veya benzerleri gibi). Alıcı-vericinin "Receiver Output Enable" ve "Driver Outputs Enable" pinleri PortC' nin 2 No'lu pinine bağlanmalıdır (bölüm sonundaki devreye bakınız).

Not: Adres 50, tüm Slave' ler için genel bir adrestir. (adres 50' yi içeren paketler tüm Slaveler tarafından alınır). İstisnai durum adresi 150 ve 169 olan Slaveler için geçerlidir; bunlar paket içerisinde kendi özel adreslerinin belirtilmiş olmasına ihtiyaç duyarlar.

Not:RS-485'i başlangıç durumuna getirmeden önce `Usart_Init` çağırılmalıdır.

Kütüphane Yordamları

```
RS485Master_Init
RS485Master_Receive
RS485Master_Send
RS485Slave_Init
RS485Slave_Receive
RS485Slave_Send
```

RS485Master_Init

Yapısı	`void Rs485master_Init(unsigned short * port, unsigned short pin);`
Tanımı	PIC MCU'yu RS-485 iletişiminde Master durumuna getirir.
Gereklilikler	HW USART birimi başlangıç durumuna getirilmelidir. `USART_Init`'e bakınız.
Örnek	`RS485Master_Init(&PORTC, 2);`

RS485Master_Receive

Yapısı	`void RS485Master_Receive(char *data);`
Tanımı	Slaveler (köleler) tarafından yollanan tüm mesajları alır. Mesaj çok baytlıdır. Bu nedenle bu yordam her bayt alımı için çağırılmalıdır (bölüm sonundaki örneğe bakınız). Bir mesajın alınması üzerine, tampon bellek aşağıdaki değerler ile doldurulur: `data[ 0..2]` mesaj, `data[ 3]` alınan mesaj byte'larının sayısı, 1–3, `data[ 4]` mesaj alındığı zaman 255'e ayarlanır, `data[ 5]` hata ortaya çıktığında 255'e ayarlanır, `data[ 6]` mesajı yollayan Slave'in adresidir. Fonksiyon otomatik olarak her mesaj alımında `data[ 4]` ve `data [ 5]` 'i ayarlar. Bu bayraklar (flags) program tarafından silinmelidirler.
Gereklilikler	Bir adres atanabilmesi için PIC MCU RS-485 iletişiminde Master durumuna getirilmelidir. `RS485Master_Init'` e bakınız.

Örnek	`unsigned short msg[ 8];`
	`...`
	`RS485Master_Receive(msg);`

RS485Master_Send

Yapısı	`void RS485Master_Send(char *data, char datalen, char address);`
Tanımı	Veriyi (`data`) RS-485 yoluyla tampon bellekten `address` parametresi ile adresi belirlenen Slave(lere) 'e gönderir. `datalen` mesajdaki byte sayısıdır. (1 <= datalen <= 3).
Gereklilikler	Bir adres atanabilmesi için PIC MCU RS-485 iletişiminde Master durumuna getirilmelidir. `RS485Master_Init`'e bakınız. Aynı anda 485 yoluyla yalnızca bir cihazın veri yolladığından emin olmak, programlayıcının sorumluluğundadır.

Örnek	`unsigned short msg[8];`
	`...`
	`RS485Master_Send(msg, 3, 0x12);`

RS485Slave_Init

Yapısı	`void Rs485Slave_Init(unsigned short * port, unsigned short pin, char address);`
Tanımı	MCU'yu, RS485 haberleşmesinde belirtilen bir adres ile bir Slave olarak başlangıç durumuna getirir. `address` parametresi ile verilen Slave adresi 0 ile 255 arasında değerler alabilir. Sadece 50 hariç. Zira bu adres tüm slave'ler için ortaktır.
Gereklilikler	HW USART birimi başlangıç durumuna getirilmelidir. `USART_Init`'e bakınız.
Örnek	`RS485Slave_Init(&PORTC, 2 ,160); /* MCU'yu address 160 ile slave olarak kur */`

RS485Slave_Receive

Yapısı	`void RS485Slave_Receive(char *data);`
Tanımı	Kendisine adreslenen mesajları alır. Mesajlar çok baytlıdır. Bu nedenle bu yordam her alınan byte için çağırılmalıdır (bölüm sonundaki örneğe bakınız). Mesaj alınması üzerine, tampon bellek aşağıdaki değer ile doldurulur: `data[ 0..2]` mesaj, `data[ 3]` alınan mesaj baytlarının sayısı, 1–3, `data[ 4]` mesaj alındığı zaman 255'e ayarlanır, `data[ 5]` hata ortaya çıktığında 255'e ayarlanır, `data[ 6]` mesaj yollanan Slave'in adresidir. Fonksiyon otomatik olarak her mesaj alımında `data[ 4]` ve `data[ 5]` 'i ayarlar. Bu bayraklar (flags) programdan silinmelidirler.
Gereklilikler	MCU RS-485 haberleşmesinde Slave olarak ayarlanmalıdır. `RS485Slave_Init`'e bakınız

Örnek	`unsigned short msg[ 8];`
	`...`
	`RS485Slave_Read(msg);`

RS485Slave_Send

Yapısı	`void RS485Slave_Send(char *data, char datalen);`
Tanımı	Veriyi (`data`) RS-485 yoluyla tampon bellekten Master'a gönderir. `datalen` mesajdaki byte'ların sayısıdır (1 <= datalen <= 3).
Gereklilikler	MCU RS-485 haberleşmesindeki Slave olarak ayarlanmalıdır. `RS485Slave_Init`'e bakınız. Aynı zamanda 485 yoluyla yalnızca bir cihazın veri yolladığından emin olmak programlayıcının sorumluluğundadır.

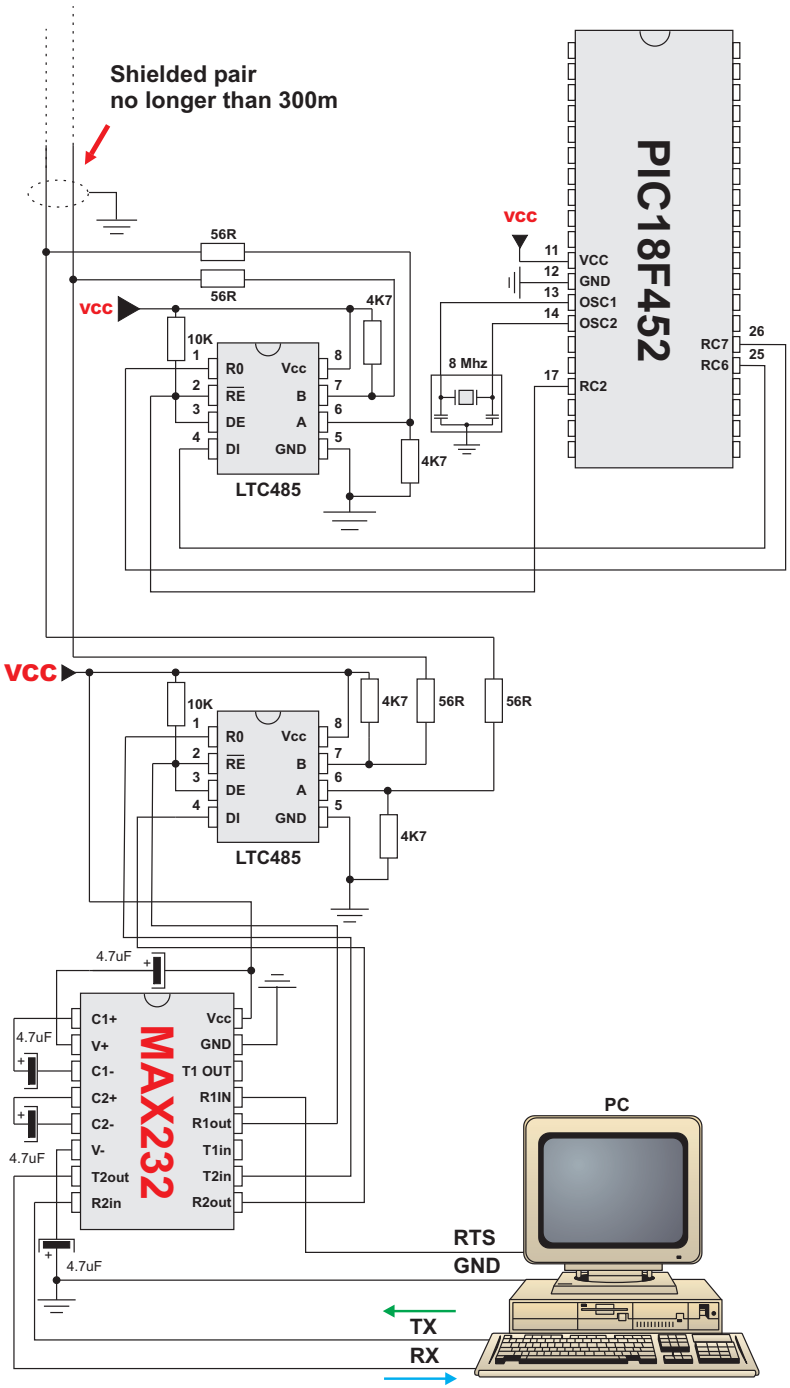
Örnek	`unsigned short msg[ 8];`
	`...`
	`RS485Slave_Send(msg, 2);`

Kütüphane Örneği

Örnek, RS-485 haberleşmesinde PIC'in Slave noktası olarak çalışmasını göstermektedir. PIC sadece kendisine adreslenen paketleri (örneğimizde adres 160'tır) ve adres 50 ile hedeflenen genel mesajları alır. Alınan veriler PortB'ye yönlendirilir ve ayrıca Master'a geri yollanır.

```
unsigned short dat[ 8];    // mesaj alma/gonderme icin tampon
char i = 0, j = 0;

void interrupt() {
  /* Her bayt RS485Slave_Read(dat) tarafindan alinir;
     Eger mesaj hatasiz alinirsa,
     data[4] 255 e kurulur */

  if (RCSTA.OERR) PORTD = 0x81;
  RS485Slave_Read(dat);
}//~

void main() {

  TRISB = 0;
  TRISD = 0;
  Usart_Init(9600);               // USART birimini kur
  RS485Slave_Init(&PORTC, 2 ,160);// MCU'yu Slave olarak kur,adres 160
  PIE1.RCIE   = 1;                // USART (RS485) yoluyla alinan bayt icin
  INTCON.PEIE = 1;                // kesmeyi (interrupt)etkin
  PIE2.TXIE   = 0;                // hale getir
  INTCON.GIE  = 1;
  PORTB  = 0;
  PORTD  = 0;
  dat[ 4] = 0;                    // Mesaj alindi bayraginin 0 oldugundan emin ol
  dat[ 5] = 0;                    // Hata bayraginin 0 oldugundan emin ol

  do {
    if (dat[ 5] ) PORTD = 0xAA;   // Hata varsa,PORTD'ye $AA gonder
    if (dat[ 4] ) {               // Mesaj alindiysa:
      dat[ 4] = 0;                //   Mesaj alındı bayragini temizle
      j = dat[ 3];                //   Alinan veri bayti sayisi
      for (i = 1; i < j; i++)
        PORTB = dat[ --i];        //   Alinan baytlari goster
      dat[ 0] ++;                 //   Alinan dat[0]'i bir arttir
      RS485Slave_Write(dat, 1);   //   Master'a geri yolla
  } while (1);
}//~!
```

Donanım Bağlantısı



Yazılımsal I2C Kütüphanesi

mikroC, I2C protokolünü yazılımsal olarak gerçekleştirecek yordamlar bulundurur. Bu yordamlar donanımdan bağımsızdırlar ve herhangi bir PIC MCU ile kullanılabilirler. YazılımsaI I2C, size PIC MCU'yu bir I2C bağlantısında Master olarak kullanmanıza olanak sağlar. Çoklu-Master modu desteklenmez.

Not: Bu kütüphane zaman-tabanlı (time-based) aktiviteleri gerçekleştirir. Bu nedenle yazılımsaI I2C kullanılırken, kesmeler (interruptlar) aktif olmamalıdır.

Kütüphane Yordamları

```
Soft_I2C_Config
Soft_I2C_Start
Soft_I2C_Read
Soft_I2C_Write
Soft_I2C_Stop
```

Soft_I2C_Config

Yapısı	`void Soft_I2C_Config(char *port, const char SDI, const char SD0, const char SCK);`
Tanımı	Yazılımsal I2C'yi yapılandırır. `port` parametresi `SDA` ve `SCL` pinlerinin bulunduğu portu belirler. `SCL` ve `SDA` parametreleri 0 ile 7 arasında olmalıdırlar ve aynı pini işaret etmemelidirler. Yazılımsal I2C kütüphanesinden bir fonksiyon çağırılmadan önce S`oft_I2C_Config` çağırılmalıdır
Örnek	`Soft_I2C_Config(&PORTB, 1, 2);`

Soft_I2C_Start

Yapısı	`void Soft_I2C_Start(void);`
Tanımı	START sinyalini yayımlar. Verinin alımından ve yollanmasından önce çağırılmalıdır .
Gereklilikler	Yazılımsal I²C bu fonksiyonun kullanımından önce yapılandırılmalıdır. `Soft_I2C_Config`'e bakınız.
Örnek	`Soft_I2C_Start();`

Soft_I2C_Read

Yapısı	`char Soft_I2C_Read(char ack);`
Dönüş	Slave'den bir byte döndürür.
Tanımı	Slave'den bir byte okur ve eğer `ack` parametresi 0 ise "kabul edilmedi sinyali" yollar. Aksi halde kabul sinyali yollar.
Gereklilikler	Bu fonksiyonun kullanılması için START sinyalinin yayımlanmış olması gerekmektedir. `Soft_I2C_Start`'a bakınız.
Örnek	`tmp = Soft_I2C_Read(0); /* Veriyi oku, kabul edilmedi sinyali yolla */`

Soft_I2C_Write

Yapısı	`char Soft_I2C_Write(char data);`
Dönüş	Eğer herhangi bir hata yoksa 0 döndürür.
Tanımı	Veri byte'ını (`data` parametresi) I²C bus'ı yoluyla gönderir.
Gereklilikler	Bu fonksiyonun kullanılması için START sinyalinin yayımlanması gerekmektedir. `Soft_I2C_Start`'a bakınız.
Örnek	`Soft_I2C_Write(0xA3);`

Soft_I2C_Stop

Yapısı	`void Soft_I2C_Stop(void);`
Tanımı	STOP sinyalini yayımlar.
Gereklilikler	Bu fonksiyonun kullanılması için START sinyalinin yayımlanması gerekmektedir. `Soft_I2C_Start`'a bakınız.
Örnek	`Soft_I2C_Stop();`

Kütüphane Örneği

```
/* Örnek yazılımsal I2C kütüphanesinin kullanımını gösterir.
PIC MCU (SCL, SDA pinleri)  PCF8583 RTC'ye bağlıdır (real-time clock).
Program RTC'ye tarih bilgisini yollar. */


void main() {

  Soft_I2C_Config(&PORTD, 4,3);  // tam master modunu başlat

  Soft_I2C_Start();              // Basla sinyalini yayınla
  Soft_I2C_Write(0xA0);          // Adres PCF8583
  Soft_I2C_Write(0);             // Adres 0'daki word'den basla (kurulum wordu)
  Soft_I2C_Write(0x80);          // Kuruluma 0x80 yaz (sayaci durdur...)
  Soft_I2C_Write(0);             // Saliseler word'une 0 yaz
  Soft_I2C_Write(0);             // Saniyeler word'une 0 yaz
  Soft_I2C_Write(0x30);          // Dakikalar word'une 0x30 yaz
  Soft_I2C_Write(0x11);          // Saatler word'une 0x11 yaz
  Soft_I2C_Write(0x30);          // Yil/tarih word'une 0x24 yaz
  Soft_I2C_Write(0x08);          // Hafta/ay'a 0x08 yaz
  Soft_I2C_Stop();               // Dur sinyalini yayinla

  Soft_I2C_Start();              // Basla sinyalini yayınla
  Soft_I2C_Write(0xA0);          // Adres PCF8530
  Soft_I2C_Write(0);             // Adres 0 daki word'den basla
  Soft_I2C_Write(0);             // Kurulum word'une 0 yaz (saymayi etkinlestir)
  Soft_I2C_Stop();               // Dur sinyalini yayinla

} //~!
```

Yazılımsal SPI Kütüphanesi

mikroC, yazılımsal olarak SPI protokolünü gerçekleştirmek için kütüphane sağlar. Bu yordamlar donanımdan bağımsızdırlar ve herhangi bir PIC MCU ile kullanılabilirler. SPI yoluyla başka aygıtlarla kolayca haberleşebilirsiniz: A/D çeviriciler, D/A çeviriciler, MAX7219, LTC1290, vs.

Kütüphane, SPI'yı master duruma yapılandırır; saat=50kHz olarak seçilir ve veri periyodun tam ortasında örneklenir. Saat darbe yok değeri "düşük" (low) olarak kabul edilir ve veri düşük kenardan yüksek kenara yükselirken iletilir.

Not: Bu kütüphane zaman tabanlı aktiviteler gerçekleştirir. Bu nedenle bu kütüphane kullanıldığında kesmeler (interrupts) etkisiz hale getirilmelidirler.

Kütüphane Yordamları

```
Soft_Spi_Config
Soft_Spi_Read
Soft_Spi_Write
```

Soft_Spi_Config

Yapısı	`void Soft_Spi_Config(char *port, const char SDI, const char SDO, const char SCK);`
Tanımı	Yazılımsal SPI'ı yapılandırır ve başlangıç durumuna getirir. `port` parametresi `SDI`, `SDO` ve `SCK` pinlerinin bulunduğu MCU portunu belirler. `SDI`, `SDO` ve `SCK` parametreleri 0 ile 7 arasında olmalıdırlar ve aynı pini işaret etmemelidir. `Soft_Spi_Config`, yazılımsal SPI kütüphanesinin diğer fonksiyonlarından önce çağırılmalıdır.
Örnek	Bu SPI'yı master durumda ayarlayacaktır. Saat= 50 kHz olarak seçilmiştir, veri zaman aralığının ortasında örneklenecektir. Saat darbesi yok (Clock idle) "düşük" (low) olarak kabul edilir ve veri düşük kenardan yüksek kenara yükselirken gönderilir. SDI pini RB1 , SDO pini RB2 , SCK pini RB3 pinidir: `Soft_Spi_Config(&PORTB, 1, 2, 3);`

Soft_Spi_Read

Yapısı	`char Soft_Spi_Read(char buffer);`
Dönüş	Alınan veriyi döndürür.
Tanımı	`buffer` içeriğini göndererek saat sağlar ve veriyi alır.
Gereklilikler	Yazılımsal SPI başlangıç durumuna getirilmeli ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Soft_Spi_Config`'e bakınız.
Örnek	`tmp = Soft_Spi_Read(buffer);`

Soft_Spi_Write

Yapısı	`void Soft_Spi_Write(char data);`
Tanımı	Veriyi (`data`) anında gönderir.
Gereklilikler	Yazılımsal SPI başlangıç durumuna getirilmeli ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Soft_Spi_Config`'e bakınız.
Örnek	`Soft_Spi_Write(1);`

Kütüphane Örneği

Bu örnek program Microchip'in 12 bit'lik D/A dönüştürücüsü olan MCP4921'in PIC MCU ile nasıl kullanıldığını göstermektedir. Bu entegre sayısal girişi (0-4095 arası) alır ve çıkış voltajına dönüştürür. Çıkış voltajı değişimi 0-Vref arasındadır. Bu örnekte D/A çevirici PORTC'ye bağlıdır. ve PIC ile SPI yoluyla haberleşir. mikroElektronika'nın DAC modülü üzerindeki referans voltajı 5V'dur. Bu örnekte, tüm DAC çözünürlük aralıkları kapsanmaktadır (0'dan 4095'e kadar). Şu anlama gelmektedir: orta aralıktan son aralığa kadar çıkış almak istiyorsanız butona yaklaşık 7 dakika basmanız gereklidir.

```
const char _CHIP_SELECT = 1, _TRUE = 0xFF;
unsigned value;

void InitMain() {
  Soft_SPI_Config(&PORTB, 4,5,3);
  TRISB &= ~(_CHIP_SELECT);             // PORTB'nin CHIP_SELECT bitini sifirla
  TRISC = 0x03;
} //~
// DAC artar (0..4095) --> cikis voltaji (0..Vref)
void DAC_Output(unsigned valueDAC) {
  char temp;
  PORTB &= ~(_CHIP_SELECT);             // PORTB'nin CHIP_SELECT bitini sifirla
  temp = (valueDAC >> 8) & 0x0F;        // Transfer icin ust bayti hazirla
  temp |= 0x30;                         //   12-bit'lik bir sayidir, bu nedenle
  Soft_Spi_Write(temp);                 //   ust baytin sadece alt yarisi kullanilir
  temp = valueDAC;                      // Transfer icin alt bayti hazirla
  Soft_Spi_Write(temp);
  PORTB |= _CHIP_SELECT;                // PORTB'nin CHIP_SELECT bitini 1 yap
} //~

void main() {
  InitMain();
  DAC_Output(2048);                     // Program basladiginda DAC
  value = 2048;                         // orta aralikta cikis verir
  while (1) {                           // Ana dongu
     if ((Button(&PORTC,0,1,1)==_TRUE)  // Test Butonu RC0'da (artma)
       && (value < 4095)) {
         value++ ;
     } else {
       if ((Button(&PORTC,1,1,1)==_TRUE)      // RC0 aktif degilse RC1'i test
         && (value > 0)) {                    // et (azalma)
         value-- ;
       }
     }
     DAC_Output(value);                 // Cikisi olustur
     Delay_ms(100);                     // Tus okuma ritmini biraz yavaslat
  }
} //~!
```

Yazılımsal UART Kütüphanesi

mikroC yazılımsal olarak UART işlevini gerçekleştirmek için bir kütüphane bulundurur. Bu yordamlar donanımdan bağımsızdırlar ve herhangi bir PIC MCU ile kullanılabilirler. RS232 protokolü yoluyla başka aygıtlarla kolayca haberleşebilirsiniz
.

Not: Bu kütüphane zaman tabanlı aktiviteler gerçekleştirir. Bu nedenle yazılımsal UART kullanıldığında kesmeler (interruptlar) etkisiz hale getirilmelidirler.

Kütüphane Yordamları

```
Soft_Uart_Init
Soft_Uart_Read
Soft_Uart_Write
```

Soft_Uart_Init

Yapısı	`void Soft_Uart_Init(unsigned short *port, unsigned short rx, unsigned short tx, unsigned short baud_rate, char inverted);`
Tanımı	Yazılımsal UART'ı başlangıç durumuna getirir. `port` parametresi `RX` ve `TX` pinlerinin olduğu MCU portunu belirtir. `RX` ve `TX` parametreleri 0 ile 7 arasında olmalıdır ve aynı pini işaret etmemelidirler; `baud_rate` istenen baud oranıdır. Maksimum baud oranı PIC'in saat frekansına ve çalışma koşullarına bağlıdır. `inverted` parametresi eğer sıfır olmayan bir değere ayarlanırsa, çıkışta terslenmiş mantık kullanılır. `Soft_Uart_Init`'in, yazılımsal UART kütüphanesinden başka fonksiyonlar kullanılmadan önce çağırılması gerekmektedir.
Örnek	`Soft_Uart_Init(&PORTB, 1, 2, 9600, 0);`

Soft_Uart_Read

Yapısı	`unsigned short Soft_Uart_Read(unsigned short *error);`
Dönüş	Alınan bir byte'ı döndürür.
Tanımı	Fonksiyon yazılımsal UART yoluyla bir byte alır. Eğer transfer işlemi sorunsuz gerçekleşmişse `error` parametresi sıfır olacaktır. Bu bir bloklanan (tıkanan) fonksiyon çağrısıdır, bu nedenle kodunuzu bu beklemeyi göz önüne alarak yazınız. Ayrıca `error` parametresini manuel olarak test etmelisiniz (aşağıdaki örneği inceleyiniz).
Gereklilikler	Yazılımsal UART başlatılmalı ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Soft_Uart_Init`'e bakınız.

Örnek	`// Burada duzgun veri alinana kadar bir dongu mevcut:`
	`error = 1`
	`do`
	`  data = Soft_Uart_Read(&error);`
	`while (error);`
	
	`// Veri saglikli bir sekilde alindi, simdi onunla calisabiliriz:`
	`if (data) { ...}`

Soft_Uart_Write

Yapısı	`void Soft_Uart_Write(char data);`
Tanımı	Fonksiyon bir byte'ı (`data`) UART yoluyla gönderir.
Gereklilikler	Yazılımsal UART başlatılmalı ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Soft_Uart_Init`'e bakınız. Gönderim boyunca yazılımsal UART'ın veri alımını yapamadığını biliniz. Bu nedenle veri kaybını önlemek için veri transfer protokolü buna göre önceden ayarlanmalıdır.

Örnek	`char some_byte = 0x0A;`
	`...`
	`Soft_Uart_Write(some_byte);`

Kütüphane Örneği

Örnek yazılımsal UART yoluyla basit veri değişimini göstermektedir. PIC MCU veriyi aldığında, aynen geri yollar. Eğer PIC, PC'ye bağlı ise (aşağıdaki örneğe bakınız), RS232 haberleşme için mikroC RS-232 haberleşme terminalinden örneği test edebilirsiniz. Bunun için derleyici menüsünden **Tools > Terminal'i** seçmelisiniz.

```
unsigned short data[ 20] ;
unsigned short ro = 0;
int i;

void main() {
  Soft_Uart_Init(&PORTC, 7, 6, 2400, 0);  // Yazilimsal UART ilk ayar
  // (8 bit, 2400 baud orani, Parite biti yok...

  while (1) {
    // Veri alma tamponunu temizle
    for (i=0;i<20;i++) {
      data[ i]  = 0;
    }

    /* Veri alma dongusunde mesaj sonu karakteri olarak sectigimiz
        '.' Karakterini alana kadar veya en fazla 20 karakter
       alana kadar bekle.   */
    for (i=0;i<20;i++) {
      data[ i]  = Soft_Uart_Read(&ro);  // Veri al

      /* Eger mesaj sonu anlaminda '.' karakteri
         alinirsa veri alma dongusunden cik.  */
      if (data[ i] =='.' ) break;
    }

    // Alinan verileri geri gonder
    for (i=0;i<20;i++) {
      // Veriyi UART ile gonder
      if ( data[ i] !=0 ) Soft_UART_Write(data[ i] );
      else break;
    }
  }
} //~!
```

Ses Kütüphanesi

mikroC uygulamalarınızda ses iletişimi yapmanızı sağlayacak bir kütüphane bulundurur. Belirtilen port'a bir basit piezo hoparlör takmanız gereklidir.

Kütüphane Yordamları

```
Sound_Init
Sound_Play
```

Sound_Init

Yapısı	`void Sound_Init(char *port, char pin);`
Tanımı	İstenilen `port` ve `pin`' den çıkış için donanımı hazırlar. `pin` parametresi 0 ile 7 arasında olmalıdır.
Örnek	`Sound_Init(&PORTB, 2);  // Sesi RB2 pinine ayarla`

Sound_Play

Yapısı	`void Sound_Play(char period_div_10, unsigned num_of_periods);`
Tanımı	Sesi istenilen `port` ve `pin`' den çalar. (`Sound_Init`'e bakınız). `period_div_10` parametresi MCU döngülerinin 10 ile bölümü ile verilen bir ses periyodudur ve bildirilen sayıda periyot boyunca (`num_of_periods`) ses üretir.
Gereklilikler	Sesi duymak için bir piezo hoparlör'e ihtiyacınız var. Ayrıca çıkış donanımının hazırlanması için `Sound_Init`'i çağırmanız gerekmektedir.
Örnek	1 KHz'de ses çıkarmak için: T = 1/f = 1ms = 1000 komut döngüsü (4MHz saat kullanıldığında). Bu bize ilk parametre olarak: 1000/10 = 100'ü verir. İkinci parametre olarak istenilen 150 periyot şu şekilde çalınır: `Sound_Play(100, 150);`

Kütüphane Örneği

Örnek, ses kütüphanesi yardımıyla tonların bir piezo hoparlör üzerinden nasıl yayınlanacağını basit bir şekilde göstermektedir. Kod, PORTB'si olan ve ADC'li PORTA'ya sahip herhangi bir PIC MCU ile kullanılabilir. Bu örnekteki ses frekansları ADC'den okunan değer ve bu değerin alt byte'ının periyot (T) olarak (f=1/T) kullanılmasıyla üretilir.

```
int adcValue;

void main() {

  PORTB  = 0;             // PORTB'yi sifirla
  TRISB  = 0;             // PORTB cikis
  INTCON = 0;             // Tum kesmeleri (interrupt) etkisiz yap
  ADCON1 = 0x82;          // VDD'yi Vref olarak;analog kanallari yapilandir
  TRISA  = 0xFF;          // PORTA giris
  Sound_Init(&PORTB, 2);         // Sesi RB2'ye ayarla

  while (1) {                    // Dongu icinde cal :
    adcValue = ADC_Read(2);      // ADC'den alt byte'i al
    Sound_Play(adcValue, 200);   // Sesi cal
  }
}
```

SPI Kütüphanesi

SPI birimi bazı PIC MCU'larda mevcuttur. mikroC Slave modunun ayarlanması ve Master modunda daha rahat çalışılması için bir kütüphane sağlar. PIC, SPI yoluyla başka aygıtlarla kolayca haberleşebilir: A/D çeviriciler, D/A çeviriciler, MAX7219, LTC1290, vs. Donanımsal olarak SPI entegre edilmiş bir PIC MCU'ya ihtiyacınız vardır (örnek olarak, PIC16F877).

Not: İki SPI'a sahip PIC mikrodenetleyiciler, mesela PIC18F8722, kullanmak istediğiniz modülü seçmenizi ister. Basitçe Spi sözcüğünün sonuna 1 veya 2 sayısını koyarak bunu gerçekleştirebilirsiniz. Örnek, `Spi2_Write();` Aynı zamanda, önceki derleyici versiyonları ile uyumlu olabilmesi ve kolay kod yönetimi için, çoklu SPI'a sahip MCU'lar için SPI kütüphanesi SPI1 kütüphanesi ile özdeştir (Örneğin: SPI işlemlerinde `SPI_Init()`'ı `SPI1_Init()` yerine kullanabilirsiniz).

Kütüphane Yordamları

```
Spi_Init
Spi_Init_Advanced
Spi_Read
Spi_Write
```

Spi_Init

Yapısı	`void Spi_Init(void);`
Tanımı	SPI'yı geçerli ayarlar ile yapılandırır ve başlangıç durumuna getirir. SPI kütüphanesinden başka fonksiyonların kullanımından önce, `Spi_Init_Advanced` veya `Spi_Init` yordamları çağırılmalıdır. Başlangıç için varsayılan ayarlar şunlardır: Master modu seçilidir, clock =Fosc/4, saatin boştayken durumu = low, alt sınırdan üst kenara geçerken veri iletimi yapılır ve giriş verisini zaman aralığının tam ortasında örneklenir. Özel konfigürasyonlar için, `Spi_Init_Advanced`'ı kullanınız.
Gereklilikler	Donanımına SPI entegre edilmiş bir PIC MCU'ya ihtiyacınız var.
Örnek	`Spi_Init();`

Spi_Init_Advanced

Yapısı	`void Spi_Init_Advanced(char master, char data_sample, char clock_idle, char transmit_edge);`

Tanımı	SPI'yı yapılandırır ve başlangıç durumuna getirir. `Spi_Init_Advanced` veya `Spi_Init`'in , SPI kütüphanesindeki diğer fonksiyonların kullanımından önce çağırılması gerekir.
	
	`mast_slav` parametresi SPI için çalışma modunu belirler ve aşağıdaki değerleri alabilir :
	`MASTER_OSC_DIV4 // Master saat=Fosc/4`
	`MASTER_OSC_DIV16 // Master saat=Fosc/16`
	`MASTER_OSC_DIV64 // Master saat=Fosc/64`
	`MASTER_TMR2 // Master saat kaynagi TMR2`
	`SLAVE_SS_ENABLE // Master Slave etkin secenegi`
	`SLAVE_SS_DIS // Master Slave pasif secenegi`
	
	`Data_sample`, verinin ne zaman örnekleneceğini belirler, şu değerleri alabilir:
	`DATA_SAMPLE_MIDDLE // Veri, araligin ortasinda orneklenir.`
	`DATA_SAMPLE_END // Veri, araligin sonunda orneklenir.`
	
	`clock_idle` parametresi saat için çalışma durumunu belirler. Şu değerleri alabilir:
	`CLK_IDLE_HIGH // Saat bos durumu=HIGH`
	`CLK_IDLE_LOW // Saat bos durumu=LOW`
	
	`transmit_edge` parametresi aşağıdaki değerleri alabilir:
	`LOW_2_HIGH // Veri gonderimi asagidan yukari cikista`
	`HIGH_2_LOW // Veri gonderimi yukaridan asagi iniste`

Gereklilikler	Donanımına SPI entegre edilmiş bir PIC MCU'ya ihtiyacınız var.
Örnek	Bu SPI'yı master moda ayarlar; saat = Fosc/4, veri zaman aralığının ortasında örneklenir, saat boş durumu = low ve veri iletimi aşağıdan yukarıya : `Spi_Init_Advanced(MASTER_OSC_DIV4, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)`

Spi_Read

Yapısı	`char Spi_Read(char buffer);`
Dönüş	Alınan veriyi döndürür.
Tanımı	`buffer` göndererek saat (clock) sağlar ve periyodun sonunda veriyi alır.
Gereklilikler	SPI başlangıç durumuna getirilmiş olmalı ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Spi_Init_Advanced` veya `Spi_Init`'e bakınız.

Örnek	`short take, buffer;`
	`...`
	`take = Spi_Read(buffer);`

Spi_Write

Yapısı	`void Spi_Write(char data);`
Tanımı	`data` baytını SSPBUF'a yazar ve gönderimi hemen başlatır.
Gereklilikler	SPI başlangıç durumuna getirilmeli ve haberleşme bu fonksiyonun kullanımından önce kurulmalıdır. `Spi_Init_Advanced` veya `Spi_Init`'e bakınız.
Örnek	`Spi_Write(1);`

Kütüphane Örneği

Kod, SPI kütüphanesi fonksiyon ve yordamlarının nasıl kullanılması gerektiğini gösterir. Konfigürasyonun şu şekilde olduğunu varsayar: max7219'un yonga seçim pini (chip select) RC1'e bağlı ve MCU'nun SDO, SDI, SCK pinleri max7219'un karşılıklı pinlerine bağlı.

```
//------------------- Fonksiyon Bildirimleri
void max7219_init1();
//------------------------------- Fonksiyon bildirimlerinin sonu

char i;

void main() {
  Spi_Init();                    // Standart yapilanma
  TRISC &= 0xFD;
  max7219_init1();               // max7219'u baslangic durumuna getir
  for (i = 1; i <= 8u; i++) {
    PORTC &= 0xFD;               // max7219'u sec
    Spi_Write(i);                // i'yi max7219'a yolla (rakamin yeri)
    Spi_Write(8 - i);            // i'yi max7219'a yolla (rakam)
    PORTC |= 2;                  // max7219'i secilmemis duruma al
  }
  TRISB = 0;
  PORTB = i;
}//~!
```

USART Kütüphanesi

Bazı PIC MCU'larda donanımsal USART birimi bulunmaktadır. mikroC USART kütüphanesi; asenkron (çift yönlü) modda rahatça çalışmayı sağlar. Böylece RS232 protokolü üzerinden başka aygıtlarla rahatça haberleşebilirsiniz. (örnek olarak: bir PC ile RS232 donanım bağlantısı için bölümün sonundaki şemaya bakınız). Bu işlem için donanımsal olarak USART içeren PIC MCU'ya ihtiyacınız vardır (örneğin: PIC16F877). Daha sonra aşağıda listelenen fonksiyonları kolaylıkla kullanabilirsiniz.
Not: USART kütüphane fonksiyonları PORTB, PORTC veya PORTG üzerinde USART birimi olan PIC MCU'ları destekler ve diğer portlarda birim bulunan PIC MCU'lar ile çalışmaz. Diğer portlarda birim kullanan PICmicro'ların örneklerini mikroC kurulum klasöründeki "Examples" klasöründe bulabilirsiniz.

Kütüphane Yordamları

```
Usart_Init
Usart_Data_Ready
Usart_Read
Usart_Write
```

Not: İki Usart'a sahip PIC mikrodenetleyiciler, mesela PIC18F8722, kullanmak istediğiniz modülü seçmenizi ister. Basitçe Usart sözcüğünün sonuna 1 veya 2 sayısını koyarak bunu gerçekleştirebilirsiniz. Örnek, `Usart2_Write();` Aynı zamanda, önceki derleyici versiyonları ile uyumlu olabilmesi ve kolay kod yönetimi için, çoklu Usart'a sahip MCU'lar için Usart kütüphanesi Usart1 kütüphanesi ile özdeştir. (Örneğin: Usart işlemlerinde `Usart_Init()`'ı `Usart1_Init()` yerine kullanabilirsiniz).

Usart_Init

Yapısı	`void Usart_Init(const long baud_rate);`
Tanımı	Donanımsal USART birimini; belirtilen baud oranı (`baud_rate`) ile başlangıç değerlerine getirir. Belirtilen osilatör frekansı Fosc ve baud oranı için entegrenin kullanım kılavuzuna bakınız. Eğer desteklenmeyen bir baud oranı seçilir ise derleyici hata verecektir. `Usart_Init`, USART kütüphanesinden başka fonksiyonların kullanımından önce çağırılmalıdır.
Gereklilikler	USART donanımlı PIC MCU'ya ihtiyacınız var.
Örnek	`Usart_Init(2400);  // Haberlesmeyi 2400 bps'e kur`

Usart_Data_Ready

Yapısı	`char Usart_Data_Ready(void);`
Dönüş	Eğer veri hazırsa fonksiyon 1 döndürür, eğer veri yoksa 0 döndürür.
Tanımı	Fonksiyon alıcı tampon belleğin okumaya hazır olup olmadığını test eder.
Gereklilikler	Donanımsal USART birimi başlangıç durumuna getirilmeli ve haberleşme, bu fonksiyonun kullanımından önce kurulmalıdır. `Usart_Init`'e bakınız.

Örnek	`int receive;`
	`...`
	`// Eger veri hazır ise onu oku:`
	`if (Usart_Data_Ready()) receive = Usart_Read;`

Usart_Read

Yapısı	`char Usart_Read(void);`
Dönüş	Alınan byte'ı döndürür. Eğer byte alınmazsa 0 döndürür.
Tanımı	Fonksiyon bir byte'ı USART yoluyla alır. Verinin hazır olup olmadığını test etmek için öncelikle `Usart_Data_Ready` fonksiyonunu çağırınız.
Gereklilikler	Donanımsal USART birimi başlangıç durumuna getirilmeli ve haberleşme, bu fonksiyonun kullanımından önce kurulmalıdır. `Usart_Init`'e bakınız.

Örnek	`int receive;`
	`...`
	`// Eger veri hazır ise onu oku:`
	`if (Usart_Data_Ready()) receive = Usart_Read;`

Usart_Write

Yapısı	`char Usart_Write(char data);`
Tanımı	Fonksiyon bir byte veriyi (`data`) USART yoluyla iletir.
Gereklilikler	Donanımsal USART birimi başlangıç durumuna getirilmeli ve haberleşme, bu fonksiyonun kullanımından önce kurulmalıdır. `Usart_Init`'e bakınız.

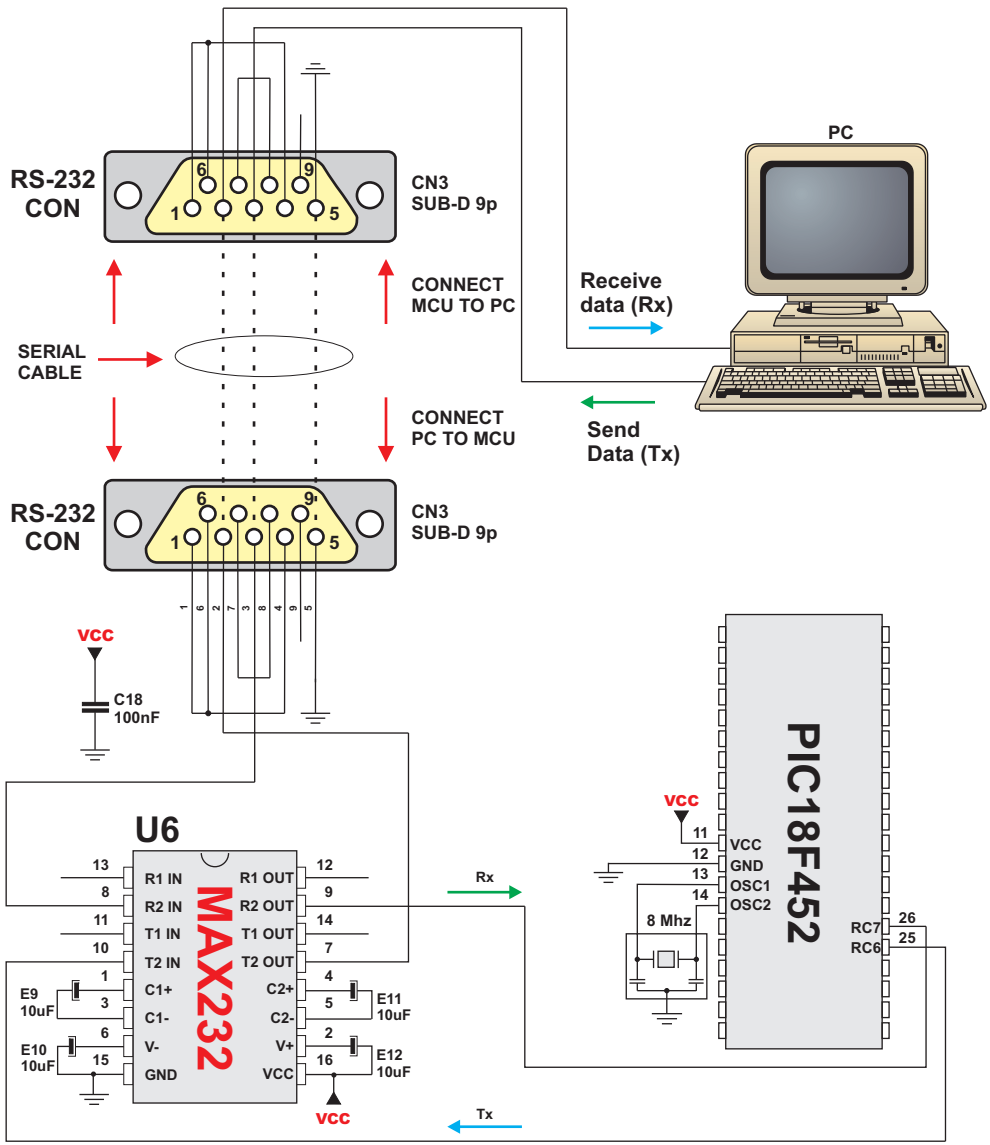
Örnek	`int Veri_Parc;`
	`...`
	`Usart_Write(Veri_Parc); // Veri parcasini USART yoluyla gonder`

Kütüphane Örneği

Örnek USART yoluyla basit veri değişimini göstermektedir. PIC aldığı veriyi hemen geri yollar. Eğer PIC, PC'ye bağlı ise (yan sayfadaki örneğe bakınız), mikroC RS232 haberleşme terminalinden örneği test edebilirsiniz. Bu işlem için derleyici menüsünden **Tools > Terminal**'i seçiniz.

```
unsigned short i;

void main() {

  // USART modülünü ayarla (8 bit veri, 2400 baud orani, parite biti yok...)
  Usart_Init(2400);

  do {
    if (Usart_Data_Ready()) {     // Eger veri alinirsa
      i = Usart_Read();           // alinan veriyi oku
      Usart_Write(i);             // ve veriyi USART yoluyla gonder
    }
  } while (1);
} //~!
```

Donanım Bağlantısı



USB HID Kütüphanesi

Universal Serial Bus (USB) bir seri iletişim yolu standardı olup birçok aygıt ile haberleşmeyi sağlar. Bilgisayarlarda, cep telefonlarında, oyun konsollarında ve PDA'larda USB arabirimi standart olarak bulunmaktadır.

mikroC, USB aracılığıyla "İnsan Arabirim Cihazları" ile çalışmak için bir kütüphane içerir. Bir İnsan Arabirim Cihazı (HID- Human Interface Device); insanlarla doğrudan etkileşimli olan ve veri girişini insandan yapan mouse, klavye, grafik tabletler, vs gibi akıllı bir aygıtdır.

Kütüphane Yordamları

```
Hid_Enable
Hid_Read
Hid_Write
Hid_Disable
```

Hid_Enable

Yapısı	`void Hid_Enable(unsigned *readbuff, unsigned *writebuff);`
Tanımı	USB HID (USB Human Interface Device - İnsan Arabirim Cihazı) haberleşmesini aktif hale getirir. `readbuff` ve `writebuff` parametreleri, HID haberleşmesinde kullanılan Okuma Tamponu ve Yazma Tamponu adresleridir. Fonksiyon USB HID kütüphanesinin diğer yordamlarının kullanımından önce çağırılmalıdır.
Örnek	`Hid_Enable(&rd, &wr);`

Hid_Read

Yapısı	`unsigned short Hid_Read(void);`
Dönüş	Anasistem'den alınan, Okuma Tampon belleğindeki karakter sayısı.
Tanımı	Anasistem'den mesajı alır ve Okuma Tampon belleğine depolar. Fonksiyon Okuma Tampon belleğindeki karakter sayısını döndürür.
Gereklilikler	USB HID, bu fonksiyon kullanılmadan önce aktif hale getirilmelidir, `Hid_Enable`'a bakınız.
Örnek	`get = Hid_Read();`

Hid_Write

Yapısı	`void Hid_Write(unsigned *writebuff, unsigned short len);`
Tanımı	Fonksiyon; `writebuff` isimli Yazma Tampon belleğinden Anasistem'e veri gönderir Write Buffer başlatma için kullanılan aynı parametredir. `len` parametresi iletilen verinin uzunluğunu belirtmelidir.
Gereklilikler	USB HID, fonksiyon kullanılmadan önce aktif hale getirilmelidir, `Hid_Enable`'a bakınız.
Örnek	`Hid_Write(&wr, len);`

Hid_Disable

Yapısı	`void Hid_Disable(void);`
Tanımı	USB HID haberleşmesini etkin olmayan duruma getirir.
Gereklilikler	USB HID, fonksiyon kullanılmadan önce aktif hale getirilmelidir, `Hid_Enable`'a bakınız.
Örnek	`Hid_Disable();`

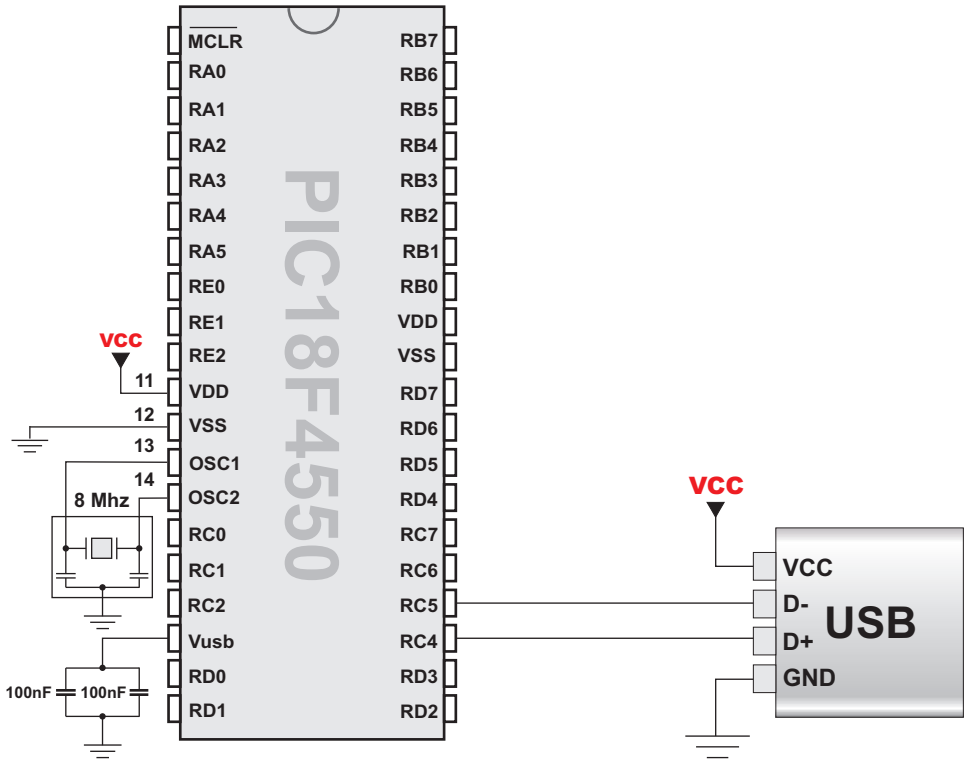
Kütüphane Örneği

Aşağıdaki örnek USB üzerinden PC'ye devamlı olarak 0 dan 255 'e kadar sayı dizisini yollamaktadır:

```
unsigned short m, k;
unsigned short userRD_buffer[ 64] ;
unsigned short userWR_buffer[ 64] ;

void interrupt() {
  asm CALL _Hid_InterruptProc
  asm nop
} //~

void Init_Main() {
  // Tum kesmeleri(interrupt) etkisiz hale getir
  // GIE, PEIE, TMR0IE, INT0IE, RBIE'yi etkisiz hale getir
  INTCON = 0;
  INTCON2 = 0xF5;
  INTCON3 = 0xC0;
  // Kesmeler uzerindeki oncelik duzeylerini etkisiz hale getir
  RCON.IPEN = 0;
  PIE1 = 0; PIE2 = 0; PIR1 = 0; PIR2 = 0;

  // Analog fonksiyonlu tum portlari dijital olarak yapilandir
  ADCON1 |= 0x0F;

  // Port'lari yapilandir
  TRISA = 0; TRISB = 0; TRISC = 0xFF; TRISD = 0xFF; TRISE = 0x07;
  LATA = 0; LATB = 0; LATC = 0; LATD = 0; LATE = 0;

  // Kullanici RAM'ini temizle
  // bloklar [00 .. 07] ( 8 x 256 = 2048 Bayt )

  asm {
    LFSR      FSR0, 0x000
    MOVLW     0x08
    CLRF      POSTINC0, 0
    CPFSEQ    FSR0H, 0
    BRA       $ - 2
  }

//devam ediyor...
```

```
    // Timer 0
    T0CON  = 0x07;
    TMR0H  = (65536-156) >> 8;
    TMR0L  = (65536-156) & 0xFF;
    INTCON.T0IE = 1;              // T0IE'yi etkin duruma getir
    T0CON.TMR0ON = 1;
}//~


/** Ana Program Yordami**/

void main() {
    Init_Main();
    Hid_Enable(&userRD_buffer, &userWR_buffer);

    do {
      for (k = 0; k < 255; k++) {
        // Gonderme tamponunu (buffer) hazirla
        userWR_buffer[0] = k;

        // Sayiyi USB yoluyla gonder
        Hid_Write(&userWR_buffer, 1);
      }
    } while (1);

    Hid_Disable();
}//~!
```

Donanım Bağlantısı



Destek Kütüphanesi (Util)

Util kütüphanesi proje geliştirmek için çeşitli kullanışlı yordamlar içermektedir.

Button

Yapısı	`char Button(char *port, char pin, char time, char active_state);`
Dönüş	0 veya 255 döndürür.
Tanımı	Fonksiyon, bir tuşa basılması üzerine oluşan bağlantı titremesini eler (elimine eder). `port` parametresi tuşun yerini belirtir. `pin` parametresi belirtilen port'un pin numarasıdır ve 0 ile 7 arasında bir değerdir. `time` parametresi mili saniye cinsinden titreme periyodudur. `active_state` parametresi 0 veya 1 olabilir ve tuşun mantıksal bir mi yoksa sıfır mı olduğunda aktif olacağını belirler.

Örnek	Örnek, RB0'a bağlı olan tuşun 1'den 0'a geçişini (tuş bırakılması) okur, sonuç olarak ise PORTD'nin evriği alınır:
	
	`do {`
	`  if (Button(&PORTB, 0, 1, 1)) oldstate = 1;`
	`  if (oldstate && Button(&PORTB, 0, 1, 0)) {`
	`  PORTD = ~PORTD;`
	`  oldstate = 0;`
	`  }`
	`} while(1);`

ANSI C Ctype Kütüphanesi

mikroC karakterleri test etme veya işleme için standart ANCI C kütüphane fonksiyonları kümesi sağlar.

Not: Tüm standart fonksiyonlar dahil edilmemiştir. Fonksiyonlar ANCI C standardına göre uygulanmıştır. Fakat, belirli fonksiyonlar PIC programlamada kolaylık sağlamak için değiştirilmişlerdir.

Kütüphane Yordamları

```
isalnum
isalpha
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit
toupper
tolower
```

isalnum

Yapısı	`char isalnum(char character);`
Tanımı	Eğer `character` Abecesayısal (alfanumerik) yani (A-Z, a-z, 0-9) kümesi içinde ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isalpha

Yapısı	`char isalpha(char character);`
Tanımı	Eğer `character` abecesel (alfabetik) yani (A-Z, a-z) kümesi içinde ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

iscntrl

Yapısı	`char iscntrl(char character);`
Tanımı	Eğer `character` bir kontrol karakteri veya 'delete'(onluk 0-31 ve 127) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isdigit

Yapısı	`char isdigit(char character);`
Tanımı	Eğer `character` rakam (0-9) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isgraph

Yapısı	`char isgraph(char character);`
Tanımı	Eğer `character` boşluk 'space'(onluk 32) dışında bir yazılabilir karakter ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

islower

Yapısı	`char islower(char character);`
Tanımı	Eğer `character` küçük harf (a-z) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isprint

Yapısı	`char isprint(char character);`
Tanımı	Eğer `character` yazılabilir (onluk sistemde ASCII 32-126 arasında) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

ispunct

Yapısı	`char ispunct(char character);`
Tanımı	Eğer `character` noktalama işareti(onluk sistemde ASCII 32-47, 58-63, 91-96, 123-126) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isspace

Yapısı	`char isspace(char character);`
Tanımı	Eğer `character` beyaz boşluk (space, CR, HT, VT, NL, FF) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isupper

Yapısı	`char isupper(char character);`
Tanımı	Eğer `character` büyük harf (A-Z) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

isxdigit

Yapısı	`char isxdigit(char character);`
Tanımı	Eğer `character` onaltılık bir rakam (0-9, A-F, a-f) ise, fonksiyon 1 döndürür. Aksi taktirde 0 döndürür.

toupper

Yapısı	`char toupper(int character);`
Tanımı	Eğer `character` küçük harf (a-z) ise, fonksiyon onun büyük harf karşılığını döndürür. Aksi taktirde giriş parametresini değişmemiş olarak döndürür.

tolower

Yapısı	`char tolower(int character);`
Tanımı	Eğer `character` büyük harf (A-Z) ise, fonksiyon onun küçük harf karşılığını döndürür. Aksi taktirde giriş parametresini değişmemiş olarak döndürür.

ANSI C Matematik Kütüphanesi

mikroC, kayan noktalı matematik işlemleri için bir standart ANSI C kütüphanesi sağlar.

Not: Fonksiyonlar ANCI C standardına göre uygulanmıştır. Fakat, belirli fonksiyonlar PIC programlamada kolaylık sağlamak için değiştirilmişlerdir.

Kütüphane Yordamları

```
acos
asin
atan
atan2
ceil
cos
cosh
exp
fabs
floor
frexp
ldexp
log
log10
modf
pow
sin
sinh
sqrt
tan
tanh
```

acos

Yapısı	`double acos(double x);`
Tanımı	Fonksiyon `x` parametresinin arkkosinüs'ünü alır, yani kosinüsü `x` olan değeri döndürür. `x` giriş parametresi -1 ve +1 arasında olmalıdır (-1 ve 1 dahil). Dönen değer radyan cinsinden ve 0 - pi arasındadır (Sıfır ve pi dahil).

asin

Yapısı	`double asin(double x);`
Tanımı	Fonksiyon `x` parametresinin arksinüs'ünü alır, yani sinüsü `x` olan değeri döndürür. `x` giriş parametresi -1 ve +1 arasında olmalıdır (-1 ve 1 dahil). Dönen değer radyan cinsinden ve –pi/2 ve pi/2 arasındadır (–pi/2 ve pi/2 dahil).

atan

Yapısı	`double atan(double x);`
Tanımı	Fonksiyon `x` parametresinin arktanjant'ını hesaplar, yani tanjantı `x` olan değeri döndürür. Dönen değer radyan cinsindendir ve –pi/2 ve pi/2 arasındadır (–pi/2 ve pi/2 dahil).

atan2

Yapısı	`double atan2(double y, double x);`
Tanımı	Bu 2 argümanlı arktanjant fonksiyonudur. '`y` bölü `x`' in arktanjantını almaya benzer, yalnız her iki bağımsız değişkenin işareti, sonucun hangi çeyrek (quadrant) te olduğunu gösterir ve x'in sıfır olmasına izin verilir. Dönen değer radyan cinsindendir ve –pi ve +pi arasındadır (–pi ve pi dahil).

ceil

Yapısı	`double ceil(double num);`
Tanımı	Fonksiyon `num` parametresinin değerini bir sonraki tam sayıya yuvarlar.

cos

Yapısı	`double cos(double x);`
Tanımı	Fonksiyon radyan cinsinden `x`'in kosinüsünü alır. Dönen değer -1 ile +1 arasındadır.

cosh

Yapısı	`double cosh(double x);`
Tanımı	Fonksiyon matematiksel olarak $(e^x+e^{-x})/2$ şeklinde belirtilen, `x`'in hiperbolik kosinüsünü alır. Eğer `x`'in değeri çok büyük ise (taşma olursa), fonksiyon başarısız olur.

exp

Yapısı	`double exp(double x);`
Tanımı	Fonksiyon e'nin (doğal logaritma tabanı) `x` üssünü alır. (yani : e^x)

fabs

Yapısı	`double fabs(double num);`
Tanımı	Fonksiyon `num`′ ın mutlak (yani pozitif) değerini alır.

floor

Yapısı	`double floor(double num);`
Tanımı	Fonksiyon `num` parametresinin değerini, altındaki en yakın tamsayıya yuvarlama yaparak döndürür.

frexp

Yapısı	`double frexp(double num, int *n);`
Tanımı	Fonksiyon kayan noktalı bir değeri (`num`), normalize edilmiş bir kesir ve 2'nin kuvveti şeklinde bir tamsayı değere ayırır. Fonksiyon normalize edilmiş kesiri döndürür ve tamsayı üs'sü de 'n' ile belirtilen işaretçinin gösterdiği yere depolar.

ldexp

Yapısı	`double ldexp(double num, int n);`
Tanımı	Fonksiyon, kayan noktalı sayının (`num`), 2'nin "n" inci kuvveti ile çarpımının sonucunu döndürür. (Dönen: $x * 2^n$).

log

Yapısı	`double log(double x);`
Tanımı	Fonksiyon x'in doğal logaritmasını alır; ($\log_e(x)$).

log10

Yapısı	`double log10(double x);`
Tanımı	Fonksiyon `x`'in 10 tabanında logaritmasını alır; ($\log_{10}(x)$).

modf

Yapısı	`double modf(double num, double *whole);`
Tanımı	Fonksiyon sayının (`num`) işaretli kesirsel bileşenini alır ve tam sayı bileşenini `whole` işaretçisi ile belirtilen değişkene yerleştirir

pow

Yapısı	`double pow(double x, double y);`
Tanımı	Fonksiyon `x`'in `y`'inci kuvetini alır; yani x^y. Eğer `x` negatif ise, fonksiyon `y`'yi otomatik olarak `unsigned long` cinsine çevirir.

sin

Yapısı	`double sin(double x);`
Tanımı	Fonksiyon `x`'in sinüsünü radyan cinsinden alır. Dönen değer -1 ile +1 arasındadır.

sinh

Yapısı	`double sinh(double x);`
Tanımı	Fonksiyon `x`'in hiperbolik sinüsünü alır. Tanımlanan matemetik işlemi: $(e^{x}-e^{-x})/2$ dir. Eğer `x` çok büyük ise (eğer taşma olursa) , fonksiyon başarısız olur.

sqrt

Yapısı	`double sqrt(double num);`
Tanımı	Fonksiyon sayının(`num`) negatif olmayan kare kökünü alır.

tan

Yapısı	`double tan(double x);`
Tanımı	Fonksiyon `x`'in radyan cinsinden tanjantını alır. Dönen değer, mikroC'in kayan noktalı sayılar için izin verilen değer aralığındadır.

tanh

Yapısı	`double tanh(double x);`
Tanımı	Fonksiyon `x`'in hiperbolik tanjantını alır. Belirtilen matematik işlemi: `sinh(x)/cosh(x)` 'dir.

ANSI C Stdlib Library

mikroC, genel kullanım için bir standart ANSI C kütüphanesi fonksiyonları seti sağlar.
Not: Tüm standart fonksiyonlar dahil edilmemiştir. Fonksiyonlar ANCI C standardına göre uygulanmıştır. Fakat, belirli fonksiyonlar PIC programlamada kolaylık sağlamak için değiştirilmişlerdir.

Kütüphane Yordamları

```
abs
atof
atoi
atol
div
ldiv
labs
max
min
rand
srand
xtoi
```

abs

Yapısı	`int abs(int num);`
Tanımı	Fonksiyon `num` değerinin mutlak değerini (yani pozitif değerini) döndürür.

atof

Yapısı	`double atof(char *s)`
Tanımı	Fonksiyon `s` giriş karakter dizisini bir `double precision` değere çevirir. Giriş karakter dizisi s kayan noktalı (floating point) literal düzene uygun olmalıdır. Başta isteğe bağlı bir beyaz boşluk (white space) olabilir. Fonksiyon, karakter dizisini, tanınmayan bir karaktere ulaşıncaya kadar, karakter karakter işleyecektir. (Buna hiçlik karakteri - null da dahildir.)

atoi

Yapısı	`int atoi(char *s);`
Tanımı	Fonksiyon `s` giriş karakter dizisini bir tamsayı (`integer`) değere çevirir ve değeri döndürür. Giriş karakter dizisi s yalnızca onluk rakamlardan oluşmalıdır. Başta isteğe bağlı bir beyaz boşluk (white space) ve + veya - işareti olabilir. Fonksiyon, karakter dizisini, tanınmayan bir karaktere ulaşıncaya kadar, karakter karakter işleyecektir. (Buna hiçlik karakteri - null da dahildir.)

atol

Yapısı	`long atol(char *s)`
Tanımı	Fonksiyon `s` giriş karakter dizisini bir uzun tamsayı (`long integer`) değere çevirir ve değeri döndürür. Giriş karakter dizisi s yalnızca onluk rakamlardan oluşmalıdır. Başta isteğe bağlı bir beyaz boşluk (white space) ve + veya - işareti olabilir. Fonksiyon, karakter dizisini, tanınmayan bir karaktere ulaşıncaya kadar, karakter karakter işleyecektir. (Buna hiçlik karakteri - null da dahildir.)

div

Yapısı	`div_t div(int numer, int denom);`
Tanımı	Fonksiyon pay olan `numer` parametresinin payda olan `denom`'a bölümünün sonucunu hesaplar. Fonksiyon bölüm (`quot`) ve kalanı (`rem`) içeren `div_t` tipinde bir yapı döndürür.

ldiv

Yapısı	`ldiv_t ldiv(long numer, long denom);`
Tanımı	Fonksiyon `div` fonksiyonuna benzerdir, ancak farklı olarak bağımsız değişkenler ve sonuç yapı üyelerinin hepsi `long` tipe sahiptir. Fonksiyon pay olan `numer`'in payda olan `denom`'a bölümünün sonucunu hesaplar. Fonksiyon bölüm (`quot`) ve kalanı (`rem`) içeren `div_t` tipinde bir yapı döndürür.

labs

Yapısı	`long labs(long num);`
Tanımı	Fonksiyon long integer olan `num` değerinin mutlak değerini döndürür.

max

Yapısı	`int max(int a, int b);`
Tanımı	Fonksiyon `a` ve `b` gibi iki tamsayıdan büyük olanını döndürür.

min

Yapısı	`int min(int a, int b);`
Tanımı	Fonksiyon `a` ve `b` gibi iki tamsayıdan küçük olanını döndürür.

rand

Yapısı	`int rand(void);`
Tanımı	Fonksiyon 0 ile 32767 arasında sözde-rastgele (pseudo-random) bir numara dizisini döndürür. `srand()` fonksiyonu ile başlama noktası belirtilmedikçe bu fonksiyon daima aynı sayı dizisini döndürür.

srand

Yapısı	`void srand(unsigned seed);`
Tanımı	Fonksiyon `seed`'i, bir önceki `rand()` fonksiyonuna yapılacak izleyen çağrılarda döndürülecek yeni sözde-rastgele sayı dizisi için başlangıç noktası olarak kullanır. Bu fonksiyon ile herhangi bir değer döndürülmez.

xtoi

Yapısı	`int xtoi(char *s);`
Tanımı	Fonksiyon onaltılık rakamlardan oluşmuş giriş karakter dizisini bir `long integer` değere çevirir. Giriş karakter dizisi `s`; yalnızca onaltılık rakamlardan oluşmalıdır. Başta isteğe bağlı bir beyaz boşluk (white space) ve + veya - işareti olabilir. Fonksiyon, karakter dizisini, tanınmayan bir karaktere ulaşıncaya kadar, karakter karakter işleyecektir. (Buna hiçlik karakteri - null da dahildir.)

ANSI C Karakter Dizisi Kütüphanesi

mikroC, karakter dizilerini (String) ve `char` dizilerini (char Array) işlemek için kullanışlı bir ANSI C kütüphanesi seti sağlar.

Not: Tüm standart fonksiyonlar dahil edilmemiştir. Fonksiyonlar ANCI C standardına göre uygulanmıştır. Fakat, belirli fonksiyonlar PIC programlamada kolaylık sağlamak için değiştirilmişlerdir.

Kütüphane Yordamları

```
memcmp
memcpy
memmove
memset
memchr
strcat
strchr
strcmp
strcpy
strlen
strncat
strncpy
strspn
strcspn
strncmp
strpbrk
strrchr
strstr
strtok
```

memcmp

Yapısı	`int *memcmp(void *s1, void *s2, int n);`
Tanımı	Fonksiyon `s1` ve `s2` ile belirtilmiş nesnelerin ilk `n` karakterlerini birbirleri ile karşılaştırır. Eğer nesneler eşit ise 0 döndürür veya eşitlik yoksa ilk farklı bulunan iki karakter arasındaki farkı döndürür (değerlendirme soldan sağa yapılır). Buna göre, eğer `s1` ile belirtilen nesne `s2` ile belirtilen nesneden büyükse sonuç sıfırdan büyük olur. Veya tersi söz konusudur.

memcpy

Yapısı	`void *memcpy(void *s1, void *s2, int n);`
Tanımı	Fonksiyon, `n` adet karakteri, `s2` ile işaretlenen nesneden, `s1` ile işaretlenen nesneye kopyalar. Nesneler örtüşemez. Fonksiyon `s1`' in değerini döndürür.

memmove

Yapısı	`void *memmove(void *s1, void *s2, int n);`
Tanımı	Fonksiyon, `n` adet karakteri, `s2` ile işaretli nesneden, `s1` işaretli nesneye kopyalar. `memcpy()`'nin aksine `s1` ve `s2` örtüşebilir. Fonksiyon `s1` in değerini döndürür.

memset

Yapısı	`void *memset(void *s, int c, int n);`
Tanımı	Fonksiyon `c` karakterinin (char'a dönüştürülmüş) değerini, `s` ile işaretlenen nesnenin ilk `n` karakterinin hepsine kopyalar. Fonksiyon `s`'in değerini döndürür.

memchr

Yapısı	`void * memchr(void *p, unsigned int n, unsigned int v);`
Tanımı	Fonksiyon, `p` adresinde başlayan bellek bölgesinin başlangıçtaki `n` byte'ı içerisinde `v` baytının ilk varlığını ve yerini bulur. Fonksiyon bu tesbitin `p` bellek adresinden olan uzaklığını döndürür veya eğer `v` bulunmazsa fonksiyon `$FFFF` döndürür. `p` parametresi için isterseniz bir bellek adresini işaret eden sayısal bir değeri (özdeğer, değişken ya da sabit) veya bir objenin adresini; örneğin `mystring` ya da `&PORTB` kullanabilirsiniz.

strcat

Yapısı	`char *strcat(char *s1, char *s2);`
Tanımı	Fonksiyon `s2` dizisini `s1` dizinine ekler. `s1` dizisinin sonundaki hiçlik karakteri (null) üzerine de yazılır. Daha sonra, bir sonlandırma hiçlik karakteri (null) sonuca eklenir. Karakter dizileri örtüşemez ve `s1` dizisi sonucu kaydedebilecek kadar yeterli alana sahip olmalıdır. Fonksiyon `s1` karakter dizisini sonuç olarak döndürür.

strchr

Yapısı	`char *strchr(char *s, char c);`
Tanımı	Fonksiyon `s` dizisinde `c` karakterinin ilk varlığını ve yerini bulur. Fonksiyon `c`'ye bir işaretçi döndürür veya `c`, `s` içerisinde bulunmuyorsa bir hiçlik işaretçisi (null pointer) döndürür. Sonlandırma hiçlik karakteri karakter dizisinin bir parçası olarak kabul edilir.

strcmp

Yapısı	`char strcmp(char *s1, char *s2);`
Tanımı	Fonksiyon `s1` ve `s2` dizilerini karşılaştırır. Eğer diziler eşit ise 0 döndürür veya ilk farklı bulunan iki karakter arasındaki farkı döndürür (değerlendirme soldan sağa yapılır). Buna göre, `s1`, `s2` den büyük ise sonuç sıfırdan büyüktür. Veya tersi söz konusudur.

strcpy

Yapısı	`char *strcpy(char *s1, char *s2);`
Tanımı	Fonksiyon `s2` dizisini `s1` dizisi içerisine kopyalar. Eğer işlem sorunsuz gerçekleşirse, fonksiyon `s1`'i döndürür. Diziler örtüşemez.

strlen

Yapısı	`unsigned strlen(char *s);`
Tanımı	Fonksiyon `s` dizisinin boyutunu döndürür (sonlandırma hiçlik karakteri (null) dizilerin uzunluklarına dahil edilmez).

strncat

Yapısı	`char *strncat(char *s1, char *s2, int n);`
Tanımı	Fonksiyon, `s2` dizisinden `s1` dizisine varsa en fazla `n` karaktere kadar ekler. `s2`'nin başlangıç karakteri `s1`'in sonundaki hiçlik (null) karakterinin üzerine yazar. Bir sonlandırma hiçlik (null) karakteri daima sonuca eklenir. Fonksiyon `s1`'i döndürür.

strncpy

Yapısı	`char *strncpy(char *s1, char *s2, int n);`
Tanımı	Fonksiyon, `s2` dizisinden `s1` dizisine varsa en fazla `n` karaktere kadar ekler. Diziler örtüşemez. Eğer `s2`, `n` karakterden kısa ise, `s1` fark kadar boşluk karakterleri ile takviye edilir. Fonksiyon `s1` dizisinin sonucunu döndürür.

strspn

Yapısı	`int strspn(char *s1, char *s2);`
Tanımı	Fonksiyon, `s1` dizisinin sadece `s2` dizisindeki karakterlerden oluşmuş maksimum başlangıç bölmesinin uzunluğunu döndürür. Dizinin sonundaki sonlandırma boşluk karakteri karşılaştırmaya dahil edilmez.

strcspn

Yapısı	`char strcspn(char * s1, char * s2);`
Tanımı	Fonksiyon, `s1` dizisinin `s2` dizisindeki karakterlerin hiçbirinden oluşmayan maksimum başlangıç bölmesinin uzunluğunu döndürür. Dizinin sonundaki sonlandırma boşluk karakteri karşılaştırmaya dahil edilmez.

strncmp

Yapısı	`int strncmp(char * s1, char * s2, char len);`
Tanımı	Fonksiyon sözlük-sırasal olarak `s1` ve `s2`′ nin ilk ‘`n`’ byte’ını karşılaştırır ve aralarındaki ilişkiyi aşağıdaki gibi gösterir: **Değer Anlamı** < 0 `s1`, `s2`’den küçüktür = 0 `s1`, `s2`’ye eşittir > 0 `s1`, `s2`’den büyüktür Döndürülen değer, karşılaştırılan dizilerdeki farklı olarak bulunan karşılıklı ilk bayt çiftinin arasındaki fark ile belirlenir. (ilk `n` byte içerisinde).

strpbrk

Yapısı	`char * strpbrk(char * s1, char * s2);`
Tanımı	Fonksiyon `s1` içerisinde `s2` dizisinin herhangi bir karakterinin ilk oluşumunu arar. Hiçlik (null) sonlandırma karakteri, aramaya dahil edilmez. Fonksiyon `s1` içerisinde ilk eşleşen karakterin indeksini döndürür. Eğer `s1` hiçbir `s2` karakterini içermezse, fonksiyon `0xFF` döndürür.

strrchr

Yapısı	`char * strrchr(char * s, char c);`
Tanımı	Fonksiyon `s` dizisi içerisinde `c` karakterinin son yerini arar. `s` dizisinin hiçlik (null) karakteri aramaya dahil edilmez. Fonksiyon `s` içerisinde son bulunan `c` karakterinin index’ini (pozisyonunu) döndürür. Eğer hiçbir eşleşen eleman bulunmazsa fonksiyon `0xFF` döndürür.

strstr

Yapısı	`char * strstr(char * s1, char * s2);`
Tanımı	Fonksiyon `s2` dizisinin `s1` dizisi içinde ilk oluşumunu arar. (hiçlik (null) sonlandırma karakteri hariç). Fonksiyon `s2`'nin `s1` içerisinde ilk bulunuşunun yer bilgisini belirten bir sayı döndürür. Eğer hiçbir dizi bulunmazsa, fonksiyon `$FF` döndürür. Eğer `s2` boş bir dizi ise, fonksiyon 0 döndürür.

strtok

Yapısı	`char * strtok(char * s1, char * s2);`
Dönüş	`strtok` fonksiyonu bir dizgeciğin ilk karakterine bir işaretçi döndürür veya dizgecik yoksa hiçlik (null) işaretçisi döndürür.
Tanımı	`strtok` fonksiyonuna peş peşe bir dizi çağrı yapıldığında `s1` dizisi bir takım dizgeciklere bölünür. Bu dizgeciklerin herbiri birbirlerinden `s2` dizisi ile belirtilen ayraçlardan biri ile ayrılmışlardır. Fonksiyon ilk çağrıldığında ilk bağımsız değişken (argüman) olarak fonksiyona `s1` dizisi verilir. Daha sonraki çağrılarda ise ilk bağımsız değişken (argüman) hiçlik (null) işaretçi olmalıdır. `s2` ile işaret edilen ayraç karakterlerinin ise bir çağrıdan diğerine değişiklik göstermelerinde bir sakınca yoktur. İlk çağrıda öncelikle ilk dizgeciği belirlemek için `s1` dizisinde `s2` dizisinde olmayan bir karakter bulunmaya çalışılır. Eğer böyle hiçbir karakter bulunamaz ise `s1` dizisinde istediğimiz bir dizgecik yok demektir ve fonksiyon hiçlik (null) işaretçi döndürür. Eğer böyle bir karakter bulunursa bu karakter aradığımız dizgeciğin ilk karakteri demektir. `strtok` fonksiyonu daha sonra `s1` dizisinde `s2` ayraçlar dizisi içerisineki ayraç karakterlerinden birini bulmaya çalışır. Eğer bu tür bir karakter bulunmazsa, oluşturulmakta olan dizgecik s1 ile işaretli dizinin sonuna kadar genişler. Ve sonradan başka dizgecikler için yapılan aramalar, bir null karakterini döndürecektir. Eğer böyle bir karakter bulunursa, üzerine bir hiçlik (null) karakteri yazılır ve mevcut dizgecik sonlandırılmış olur. Bu esnada `strtok` fonksiyonu içsel olarak bir sonraki karakteri gösteren bir işaretçi saklar ki bir sonraki dizgecik aranırken bu karakterden başlanır. Fonksiyonun takip eden her çağrısında, argüman olarak hiçlik işaretleyicisi kullanılırsa, araştırmaya daha önce sakladığımız işaretçiden itibaren başlanır ve arama yukarıda anlatıldığı şekilde yapılır.

Örnek	`char x[10] ;`
	`void main(){`
	`  strcpy(x, strtok("mikroE1", "Ek"));  // x dizisi "mi" olur.`
	`  strcpy(x, strtok(0, "kE"));  // x dizisi "ro" olur.`
	`}`

Dönüşümler (Conversions) Kütüphanesi

mikroC Dönüşümler Kütüphanesi, sayılardan dizilere (numerals to strings) dönüşüm ve İkili Kodlanmış Onlu (BCD) Sistem/Onlu Sistem dönüşümleri sağlayan yordamları içerir.

Kütüphane Yordamları

Aşağıdaki yordamları kullanarak sayısal değerlerin metin gösterimini elde edebilirsiniz:

```
ByteToStr
ShortToStr
WordToStr
IntToStr
LongToStr
FloatToStr
```

Aşağıdaki fonksiyonlar onluk değeri İkili Kodlanmış Onlu'ya (BCD'ye) çevirir veya tersi işlem yaparlar:

```
Bcd2Dec
Dec2Bcd
Bcd2Dec16
Dec2Bcd16
```

ByteToStr

Yapısı	`void ByteToStr(unsigned short number, char *output);`
Tanımı	Bu işlev küçük işaretsiz bir sayının (`number`) karakter dizisi karşılığını (`output`) oluşturur (0x100 den küçük sayılar için). Çıkış dizisi 3 karakter sabit boyutundadır. Solda kalan pozisyonlar (eğer varsa) boşluklar ile doldurulur.

Örnek	`unsigned short t = 24;`
	`char txt[4];`
	`//...`
	`ByteToStr(t, txt);  // txt " 24" olur (Basta bir bosluk var)`

ShortToStr

Yapısı	`void ShortToStr(short number, char *output);`
Tanımı	Fonksiyon küçük işaretli bir sayının (`number`) karakter dizisi karşılığını (`output`) oluşturur (0x100 den küçük sayılar). Çıkış dizisi 4 karakter sabit boyutundadır . Solda kalan pozisyonlar (eğer varsa) boşluklar ile doldurulur.

Örnek	`short t = -24;`
	`char txt[5];`
	`//...`
	`ByteToStr(t, txt); // txt " -24" olur (Basta bir bosluk var)`

WordToStr

Yapısı	`void WordToStr(unsigned number, char *output);`
Tanımı	Fonksiyon işaretsiz sayıdan (`number`) karakter dizisi karşılığını (`output`) oluşturur Çıkış dizisi 5 karakter sabit boyutundadır. Solda kalan pozisyonlar (eğer varsa) boşluklar ile doldurulur .

Örnek	`unsigned t = 437;`
	`char txt[6];`
	`//...`
	`WordToStr(t, txt); // txt "  437" olur (Iki bosluk var)`

IntToStr

Yapısı	`void IntToStr(int number, char *output);`
Tanımı	Fonksiyon küçük işaretli bir sayıdan (`number`) karakter dizisi karşılığını (`output`) oluşturur. Çıkış dizisi 6 karakter sabit boyutundadır. Solda kalan pozisyonlar (eğer varsa) boşluklar ile doldurulur.

Örnek	`int j = -4220;`
	`char txt[7];`
	`//...`
	`IntToStr(j, txt); // txt " -4220" olur (bir bosluk var)`

LongToStr

Yapısı	`void LongToStr(long number, char *output);`
Tanımı	Fonksiyon büyük işaretli sayıdan (`number`) (`long` tipinde sayısal değer) karakter dizisi karşılığını (`output`) oluşturur. Çıkış dizisi 11 karakter sabit boyutundadır. Solda kalan pozisyonlar (eğer varsa) boşluklar ile doldurulur.

Örnek	`long jj = -3700000;`
	`char txt[12];`
	`//...`
	`LongToStr(jj, txt);  // txt "   -3700000" olur (Uc bosluk var)`

FloatToStr

Yapısı	`void FloatToStr(float number, char *output);`
Tanımı	Fonksiyon kayan noktalı sayıdan (`number`) karakter dizisi karşılığını (`output`) oluşturur. Çıkış dizisi `number`'ın normalize edilmiş halini (yani ondalık kısım 0 ve 1 arasında) ilk pozisyondaki işaretiyle birlikte içerir. Ondalık kısım 6 rakamla sabitlenmiş formdadır. Gösterim olarak `0.ddddd` formunu verebiliriz. Noktadan sonra daima 5 rakam vardır. Sayının yazıya dönüştürülmüş formunun koyulacağı çıkış dizisi (`output`) en az 13 karakter boyutuna sahip olmalıdır.

Örnek	`float ff = -374.2;`
	`char txt[13];`
	`//...`
	`FloatToStr(ff, txt);  // txt "-0.37420e3" olur`

Bcd2Dec

Yapısı	`unsigned short Bcd2Dec(unsigned short bcdnum);`
Dönüş	BCD'den dönüştürülmüş Onluk değeri döndürür.
Tanımı	8-bitlik İkili Kodlanmış Onlu (BCD) değeri (`bcdnum`) Onluk eşleniğine çevirir.

Örnek	`unsigned short a;`
	`...`
	`a = Bcd2Dec(0x52);  // = 52`

Dec2Bcd

Yapısı	`unsigned short Dec2Bcd(unsigned short decnum);`
Dönüş	Onluk'tan dönüştürülmüş BCD değeri döndürür.
Tanımı	8-bitlik Onluk değeri (`decnum`) İkili Kodlanmış Onlu (BCD) eşleniğine çevirir.

Örnek	`unsigned short a;`
	`...`
	`a = Dec2Bcd(52);  // = 0x52`

Bcd2Dec16

Yapısı	`unsigned Bcd2Dec16(unsigned bcdnum);`
Dönüş	BCD'den dönüştürülmüş olan Onluk değeri döndürür.
Tanımı	16-bitlik İkili Kodlanmış Onlu (BCD) değeri (`bcdnum`) Onluk eşleniğine çevirir.

Örnek	`unsigned a;`
	`...`
	`a = Bcd2Dec16(1234); // = 0001001000110100 ikilik = 4660 onluk`

Dec2Bcd16

Yapısı	`unsigned Dec2Bcd(unsigned decnum);`
Dönüş	Onluk'tan dönüştürülmüş BCD değeri döndürür.
Tanımı	16-bitlik Onluk değeri (`decnum`) BCD'ye çevirir

Örnek	`unsigned a;`
	`...`
	`a = Dec2Bcd16(4660); // = 0001 0010 0011 0100 ayirinca = 1234 BCD`

Trigonometri Kütüphanesi

mikroC temel trigonometri fonksiyonlarını gerçekleştirmiştir. Bu fonksiyonlar başvuru tablosu olarak gerçeklenmişlerdir ve sonucu tamsayı olarak, 1000 ile çarpılmış ve yukarı yuvarlanmış olarak döndürürler.

Kütüphane Yordamları

```
SinE3
CosE3
```

SinE3

Yapısı	`int SinE3(unsigned angle_deg);`
Dönüş	Fonksiyon giriş parametresinin sinüsünü, 1000 (1E3) ile çarpılmış ve daha büyük en yakın tam sayıya yuvarlatılmış olarak döndürür. Dönen değerlerin aralığı -1000 ile 1000 arasındadır.
Tanımı	Fonksiyon açıyı derece cinsinden gösteren `angle_deg` parametresini girdi olarak alır ve onun sinüsünü 1000 ile çarpılmış ve en yakındaki, daha büyük tamsayıya yuvarlatılmış olarak döndürür. Fonksiyon bir başvuru tablosundan değer okuyarak döndürülecek sonucu hazırlar; maximum hata ±1 olarak elde edilir. Formül verecek olursak: *`sonuc = round_up(sin(angle_deg)*1000)`*
Örnek	`res = SinE3(45);  // sonuc = 707`

CosE3

Yapısı	`int CosE3(unsigned angle_deg);`
Dönüş	Fonksiyon giriş parametresinin cosinüsünü, 1000 (1E3) ile çarpılmış ve daha büyük en yakın tam sayıya yuvarlatılmış olarak döndürür. Dönen değerlerin aralığı -1000 ile 1000 arasındadır.
Tanımı	Fonksiyon açıyı derece cinsinden gösteren `angle_deg` parametresini girdi olarak alır ve onun kosinüsünü 1000 ile çarpılmış ve en yakındaki, daha büyük tamsayıya yuvarlatılmış olarak döndürür. Fonksiyon bir başvuru tablosundan değer okuyarak döndürülecek sonucu hazırlar; maximum hata ±1 olarak elde edilir. Formül verecek olursak: *`sonuç = round_up(cos(angle_deg)*1000)`*
Örnek	`res = CosE3(196);  // sonuc = -193`

Sprint Karakter Dizisi (String) Oluşturma Kütüphanesi

Kütüphane sprint fonksiyonları içindir.
Not: ANCI C standardına ek olarak mikroC, daha az RAM ve ROM alan sprinti, sprintlt gibi daha sınırlı işlevleri sağlar. Daha az ROM ve RAM kullanmalarından dolayı bu işlevler PIC için bazı durumlarda çok kullanışlı olabilir.

Kütüphane Yordamları

sprintf
sprintl
sprinti

sprintf

Tanımı: Fonksiyon karakter dizisi ve numerik değerleri biçimlendirir ve sonuçta oluşan karakter dizisini *buffer* dizisi içinde depolar.

Not: biçimlendirme dizisi CONST alanı içinde olmalıdır. sprintf fonksiyonu P12 ve P16 PIC MCU ailesi için desteklenmez.

fmtstr bağımsız değişkeni bir biçimlendirme dizisidir ve karakterlerden, kaçış dizilerinden ve biçimlendirme özelliklerinden oluşmuş olabilir. Sıradan karakterler ve kaçış dizileri *buffer* karakter dizisi içine derleyicinin yorumlama sırasına göre yerleştirilirler. Biçimlendirme tarifleri daima bir yüzde '%' işareti ile başlar ve fonksiyon çağrısında ek bağımsız değişkenlerin kullanımına ihtiyaç duyar.

Biçimlendirme dizisi soldan sağa okunur. İlk biçimlendirme tarifi fmtstr'den sonraki ilk bağımsız değişken içindir ve onu biçimlendirme tarifini kullanarak dönüştürür ve çıktı olarak verir. İkinci biçimlendirme tarifi de fmtstr'den sonra ikinci bağımsız değişken içindir ve bu işlem aynı şekilde devam eder. Eğer biçimlendirme tariflerinden daha fazla sayıda bağımsız değişken varsa, fazlalık bağımsız değişkenler ihmal edilir. Eğer biçimlendirme tariflerinden daha az bağımsız değişken varsa sonucun ne olacağı kestirilemez. Biçimlendirme tarifleri formatı:

% [bayraklar] [genişlik] [.duyarlık] [{ h | l | L }] dönüştürme_tipi

Biçimlendirme tarifi içindeki her alan, bir özel biçim seçimini belirten tek bir karakter veya bir sayı olabilir. *dönüştürme_tipi* alanındaki karakter bağımsız değişkenin bir karakter mi, bir karakter dizisi mi, bir sayı mı yoksa bir işaretçi mi olarak yorumlanacağını belirtir. Sonraki tabloda bunu görebilirsiniz.

Dönüştürme tipi	Bağ. değ. tipi	Çıkış biçimi
d	int	İşaretli onluk sayı
u	unsigned int	İşaretsiz onluk sayı
o	unsigned int	işaretsiz sekizlik sayı
x	unsigned int	0123456789ABCEDF'yı kullanan işaretsiz onaltılık sayı
X	double	[-] dddd.dddd formatını kullanan kayan noktalı sayı
e	double	[-] d.dddde[-] dd formatını kullanan kayan noktalı sayı
E	double	[-] d.ddddE[-] dd formatını kullanan kayan noktalı sayı
g	double	e veya f formatını kullanan kayan noktalı sayı (belirtilen değer ve hassaslık için hangi gösterim daha kısa ise)
c	int	int işaretsiz bir char'a dönüştürülür, ve sonuç karakter yazılır.
s	char *	Hiçlik (null) karakteri ile sonlandırılan karakter dizisi
p	void *	İşaretçi değeri, X formatı kullanılır.
%	none	Bir '%' işareti yazılır. Hiçbir bağımsız değişken çevrilmez. Tam dönüşüm tarifi %% olmalıdır.

Flags (bayraklar) alanında tek bir karakter çıkışın doğru yapılmasına yardımcı olur. Örneğin +/- işaretlerinin ve boşlukların yazılması, onluk nokta, sekizlik ve onaltılık ön-ekler gibi. Aşağıdaki tabloda bunlar listelenmiştir.

Bayraklar	**Anlamı**
-	Belirtilen alan genişliğinde çıkışı sola yaslandırır.
+	Eğer çıkış işaretli ise çıkış değerinin önüne + veya - koyar.
boşluk (' ')	Eğer çıkış pozitif işaretli bir değer ise çıkış değerinin önüne bir boşluk koyar. Aksi durumda hiçbir boşluk değerin önüne konmaz.
#	Sırasıyla o, x ve X alan tipleri ile kullanıldığında, sıfır olmayan bir çıkış değerinin önüne 0, 0x veya 0X koyar. e, E, f ve G alan tipleri ile kullanıldığında, # bayrak bir onluk noktayı içermesi için çıkışı güçlendirir. Diğer tüm durumlarda # bayrağı ihmal edilir.
*	Biçim özellikleri ihmal edilir.

Genişlik (width) alanı, yazılan karakterlerin minimum sayısını belirten ve negatif olmayan bir sayıdır. Eğer çıkış değeri içerisindeki karakterlerin sayısı *genişlik*'ten küçük ise, sola veya (- bayrağı kullanılmışsa) sağa minimum genişliği doldurmak için boşluklar eklenir. Eğer genişlik önünde bir 0 öneki varsa boşluklar yerine sıfırlar doldurulur. *Genişlik* alanı hiçbir zaman bir alanda biçimlenecek veriyi kırpmaz ve eğer çıkış değerinin uzunluğu belirtilen *genişliği* aşıyorsa tüm karakterler yazılır.

Duyarlık alanı, yazılacak karakter sayısını, anlamlı rakamların sayısını veya onluk noktadan sonraki yerleşim sayısını belirten negatif olmayan bir sayıdır. Duyarlık alanı, aşağıdaki tabloda belirtildiği gibi kayan noktalı sayıdan dolayı kırpmaya veya çıkış değerinin yuvarlanmasına neden olabilir.

Bayraklar	Duyarlık (precision) alanının anlamı
d, u, o, x, X	Duyarlık alanı çıkış değerine eklenebilecek minimum sayıda rakamın belirtildiği yerdir. Bağımsız değişken içerisindeki rakamların sayısı duyarlık alanı içerisinde belirtildiğinden daha uzun ise bu rakamlar kırpılmazlar. Eğer bağımsız değişken içerisindeki rakamların sayısı duyarlık alanındaki rakamlardan az ise, çıkış değeri sıfırlar ile doldurulur.
f	Duyarlık alanı , onlu noktanın sağındaki rakamların sayısını belirttiğiniz yerdir. En sondaki rakam yuvarlanır.
e, E	Duyarlık alanı , onlu noktanın sağındaki rakamların sayısını belirttiğiniz yerdir. En sondaki rakam yuvarlanır.
g	Duyarlık alanı , çıkış değeri içerisinde maksimum anlamlı karakter sayısının belirtildiği yerdir
c, C	Duyarlık alanının bu alan tipleri üzerinde hiçbir etkisi yoktur.
s	Duyarlık alanı , çıkış değeri içerisinde maksimum karakter sayısının belirtildiği yerdir. Fazlalık olan karakterler çıkışa koyulmazlar.

Seçimlik karakterler olan h ve l (veya L)'nin uygun olanı, yukarıda sözü geçen *d, i, u, o, x*, ve *X* tamsayı tiplerinin kısa veya uzun (short or long) çeşitlerini belirtmek için, *dönüştürme_tipi*'nin hemen önüne getirilebilir.

Bağımsız değişken tipinin biçim tarifiyle uyuştuğuna emin olunuz. Tip çevrimlerini (type casts) uygun tipin *sprintf*'e geçildiğinden emin olmak için kullanabilirsiniz.

sprintl

Yapısı	`int sprintl (`
	`  char *buffer,  /* Depolama tamponu */`
	`  const char *fmtstr,  /* Bicimlendirme (format) dizisi */`
	`  ... );  /* Ek bagimsiz degiskenler */`

Dönüş	Fonksiyon tampona gerçekten yazılan karakterlerin sayısını döndürür.
Tanımı	Kayan noktalı sayıları desteklememesinin dışında, sprintf ile aynıdır.

sprinti

Yapısı	`int sprinti (`
	`  char *buffer,  /* Depolama tamponu */`
	`  const char *fmtstr,  /* Bicimlendirme (format) dizisi */`
	`  ... );  /* Ek bagimsiz degiskenler */`

Dönüş	Fonksiyon tampona gerçekten yazılan karakterlerin sayısını döndürür.
Tanımı	Uzun tamsayı tipi sayıları desteklememesinin dışında, sprintl ile aynıdır.

SPI Grafik LCD kütüphanesi

mikroC, 128x64'lük grafik LCD'yi SPI yoluyla çalıştırmak için bir kütüphane sağlamaktadır. Bu yordamlar çok kullanılan 128x64 GLCD'ler ile çalışmaktadırlar (Örnek : samsung ks0108).

Önemli!: SPI kütüphane yordamlarını kullandığınız zaman, çalışan SPI modülünü `Spi_Glcd_Init` fonksiyonu içerisinde SPI1 veya SPI2 olarak belirtmeniz gerekir.

Not: Aşağıdaki kütüphane yordam veya fonksiyonlarından herhangi birini kullanmadan önce, GLCD'li portunun çıkış olarak (output) seçildiğinden emin olunuz.

Not: SPI GLCD başlangıç durumuna getirilmeden önce `Spi_Init` çağrılmalıdır.

Kütüphane Yordamları

Temel Yordamlar:

```
Spi_Glcd_Init
Spi_Glcd_Set_Side
Spi_Glcd_Set_Page
Spi_Glcd_Set_X
Spi_Glcd_Read_Data
Spi_Glcd_Write_Data
```

Gelişmiş Yordamlar:

```
Spi_Glcd_Fill
Spi_Glcd_Dot
Spi_Glcd_Line
Spi_Glcd_V_Line
Spi_Glcd_H_Line
Spi_Glcd_Rectangle
Spi_Glcd_Box
Spi_Glcd_Circle
Spi_Glcd_Set_Font
Spi_Glcd_Write_Char
Spi_Glcd_Write_Text
Spi_Glcd_Image
```

Spi_Glcd_Init

Yapısı	`void SPI_Glcd_Init(char DeviceAddress, unsigned int * rstport, unsigned int rstpin, unsigned int * csport, unsigned int cspin);`
Tanımı	128x64'lük GLCD'yi SPI yoluyla başlangıç durumuna getirir. `RstPort` ve `RstPin` parametreleri spi genişleticinin reset pinine bağlı MCU port ve pinini ayarlar. `CSPort` ve `CSPin` parametreleri spi genişleticinin CS pinine bağlı MCU port ve pinini ayarlar. `DeviceAddress` parametresi Spi genişleticinin adresidir (Spi genişletici üzerindeki A0, A1 ve A2 pinlerinin VCC veya GND'a bağlanması ile oluşturulmuş donanımsal adres.)
Gereklilikler	**Not**: SPI GLCD başlangıç durumuna getirilmeden önce `Spi_Init` çağırılmalıdır. Bu yordam SPI GLCD kütüphanesinin diğer yordamlarının kullanılmasından önce çağırılmalıdır.
Örnek	`Spi_Glcd_Init(0,&PORTC, 0, &PORTC, 1);`

Spi_Glcd_Set_Side

Yapısı	`void SPI_Glcd_Set_Side(char x_pos);`
Tanımı	GLCD'nin sağ veya sol yarısını seçer. `x` parametresi seçilen yarıyı belirler: 0'dan 63'e kadar olan değerler sol tarafı, 64'ten büyük değerler sağ tarafı belirtir. `Spi_Glcd_Set_Side`, `Spi_Glcd_Set_X` ve `Spi_Glcd_Set_Page` fonksiyonlarının kullanılması ile GLCD üzerinde tam bir pozisyon belirtilebilir. Daha sonra, `Spi_Glcd_Write_Data` veya `Spi_Glcd_Read_Data` fonksiyonlarını bu pozisyondayken kullanabilirsiniz.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Select_Side(0);` `Spi_Glcd_Select_Side(10);`

Spi_Glcd_Set_Page

Yapısı	`void Spi_Glcd_Set_Page(char page);`
Tanımı	GLCD'nin sayfasını (page) seçer, teknik olarak sayfa display üzerindeki bir satır demektir; `page` parametresi 0 dan 7'ye kadar bir değer alabilir.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Set_Page(5);  // Pozisyon olarak 5. satir seciliyor.`

Spi_Glcd_Set_X

Yapısı	`void SPI_Glcd_Set_X(char x_pos);`
Tanımı	Verilen sayfada (satırda) GLCD'nin sol kenarından `x` nokta kadar uzağı seçer.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Set_X(25);  // Pozisyon olarak soldan 25. noktayi sec.`

Spi_Glcd_Read_Data

Yapısı	`char Spi_Glcd_Read_Data();`
Dönüş	GLCD hafızasından bir word okur.
Tanımı	GLCD belleğinin geçerli konumundan veri okur. `Spi_Glcd_Set_Side`, `Spi_Glcd_Set_X` ve `Spi_Glcd_Set_Page` fonksiyonlarının kullanılması ile GLCD üzerinde tam bir pozisyon belirtilebilir. Daha sonra, `Spi_Glcd_Write_Data` veya `Spi_Glcd_Read_Data` fonksiyonlarını bu pozisyondayken kullanabilirsiniz.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`tmp = Spi_Glcd_Read_Data;`

Spi_Glcd_Write_Data

Yapısı	`void Spi_Glcd_Write_Data(char data);`
Tanımı	GLCD belleğinin geçerli konumuna veri (`data`) yazar ve bir sonraki konuma geçer.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Write_Data(data)`

Spi_Glcd_Fill

Yapısı	`void Spi_Glcd_Fill(char pattern);`
Tanımı	GLCD belleğini `pattern` byte örnekleri ile doldurur. GLCD ekranını temizlemek için, `Spi_Glcd_Fill(0)`'ı kullanın; ekranı komple noktalarla doldurmak için, `Spi_Glcd_Fill($FF)`'i kullanınız.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Fill(0);  // Ekrani temizle`

Spi_Glcd_Dot

Yapısı	`void Spi_Glcd_Dot(char x_pos, char y_pos, char color);`
Tanımı	GLCD üzerinde `(x,y)` koordinatına bir nokta çizer. `color` parametresi noktanın durumunu belirler: 0: noktayı temizler, 1: bir nokta koyar ve 2: noktanın durumunu tersine çevirir.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Dot(0, 0, 2);  /* Sol ust kosedeki noktanin durumunu ters cevirir. */`

Spi_Glcd_Line

Yapısı	`void SPI_Glcd_Line(int x_start, int y_start, int x_end, int y_end, char color);`
Tanımı	GLCD üzerinde (`x1,y1`)'den (`x2,y2`)'e bir çizgi çizer. `color` parametresi noktaların durumunu belirtir; 0 :boş satır, 1:dolu satır ve 2 : satırın her noktasını tersine çevirir (akıllı satır).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Line(0, 63, 50, 0, 2);`

Spi_Glcd_V_Line

Yapısı	`void Spi_Glcd_V_Line(char y_start, char y_end, char x_pos, char color);`
Tanımı	GLCD üzerinde (`x,y1`)'den (`x,y2`)'e bir dikey çizgi çizer. `color` parametresi noktaların durumunu belirtir; 0 :boş satır, 1:dolu satır ve 2 : satırın her noktasını tersine çevirir (akıllı satır).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_V_Line(0, 63, 0, 1);`

Spi_Glcd_H_Line

Yapısı	`void Spi_Glcd_H_Line(char x_start, char x_end, char y_pos, char color);`
Tanımı	GLCD üzerinde (`x1,y`)'den (`x2,y`)'e konumuna bir yatay çizgi çizer. `color` parametresi noktaların durumunu belirtir; 0 :boş satır, 1:dolu satır ve 2 : satırın her noktasını tersine çevirir (akıllı satır).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_H_Line(0, 127, 0, 1);`

Spi_Glcd_Rectangle

Yapısı	`void Spi_Glcd_Rectangle(char x_upper_left, char y_upper_left, char x_bottom_right, char y_bottom_right, char color);`
Tanımı	GLCD üzerine bir dikdörtgen çizer. (`x1,y1`) parametreleri sol üst köşeyi, (`x2,y2`) sağ alt köşeyi ayarlar. `color` parametresi kenarları belirler; 0 :boş kenar (nokta yok), 1:dolu kenar (nokta var) ve 2 : akıllı (smart) kenar (her noktayı tersine çevirir).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Rectangle(10, 0, 30, 35, 1);`

Spi_Glcd_Box

Yapısı	`void Spi_Glcd_Box(char x_upper_left, char y_upper_left, char x_bottom_right, char y_bottom_right, char color);`
Tanımı	GLCD üzerine bir kutu çizer. (`x1,y1`) parametreleri sol üst köşeyi, (`x2,y2`) sağ alt köşeyi ayarlar. `color` parametresi kutunun dolum durumunu belirler: 0 :beyaz kutu (nokta yok), 1: dolu kutu (nokta var) ve 2: tersine dönmüş kutu (her noktayı tersine çevirir).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Box(10, 0, 30, 35, 1);`

Spi_Glcd_Circle

Yapısı	`void Spi_Glcd_Circle(int x_center, int y_center, int radius, char color);`
Tanımı	GLCD üzerine (`x,y`) merkezli `radius` yarıçaplı bir çember çizer. `color` parametresi çember çizgilerini belirler: 0:boş çember (nokta yok), 1:dolu çember (nokta var) ve 2: akıllı (smart) çember (her noktayı tersine çevirir).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Circle(63, 31, 25, 1);`

Spi_Glcd_Set_Font

Yapısı	`void SPI_Glcd_Set_Font(const char * activeFont, char aFontWidth, char aFontHeight, unsigned int aFontOffs);`
Tanımı	`Spi_Glcd_Write_Char` ve `Spi_Glcd_Write_Text` gibi ekran metin yordamları için font'u ayarlar. Font'un bir `byte` dizisi gibi düzenlenmesi gerekir. `activeFont` parametresi font'un adresini belirler. `aFont_width` ve `aFont_height` parametreleri karakterin nokta cinsinden genişliğini ve yüksekliğini gösterir. Font genişliği 128 noktayı geçmemeli, font'un yüksekliği de 8 noktayı geçmemelidir. `font_offset` parametresi font'un hangi ASCII karakterden başladığını belirler. Kütüphanedeki demo fontlar 32 offsete sahiptirler, yani boşluk (space) ile başlarlar. Eğer hiçbir font belirtilmemiş ise; `Spi_Glcd_Write_Char` ve `Spi_Glcd_Write_Text` yordamları kütüphane ile verilmiş olan 5x8 font'u kullanırlar. "`GLCD_Fonts.c`" dosyasında verilmiş rehber bilgilerle kendi fontunuzu oluşturabilirsiniz. Bu dosya GLCD için geçerli fontları içerir ve kurulum klasörüne yerleştirilmiştir; **Extra samples > GLCD**.
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`// space(32)ile baslayan ozel 5x7 font'u ("myfont") kullan:` `Spi_Glcd_Set_Font(myfont, 5, 7, 32);`

Spi_Glcd_Write_Char

Yapısı	`void SPI_Glcd_Write_Char(char chr1, char x_pos, char page_num, char color);`
Tanımı	Sayfaya (0-7 arasındaki 8 GLCD satırından birine) ekranın sol kenarından x nokta uzağa karakteri yazar. `color` parametresi doldurmayı belirler: 0 :beyaz karakter (nokta yok), 1:dolu karakter (nokta var) ve 2 : akıllı (smart) karakter (her noktayı tersine çevirir).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız. Ekranın fontunu belirlemek için `Spi_Glcd_Set_Font` yordamını kullanınız. Eğer herhangi bir font seçilmemiş ise, kütüphane tarafından sağlanan 5x8 font kullanılır.
Örnek	`Spi_Glcd_Write_Char('C', 0, 0, 1);`

Spi_Glcd_Write_Text

Yapısı	`void SPI_Glcd_Write_Text(char text[], char x_pos, char page_num, char color);`
Tanımı	Sayfaya, sol kenardan `x` nokta uzağa ve 0-7 arasında 8 GLCD satırından seçilen birisine verilen metni (`text`) yazar.`color` parametresi noktaların durumunu belirtir; 0:beyaz karakter (nokta yok) , 1:dolu karakter (nokta var) ve 2: akıllı karakter (her nokta tersine çevrilmiştir).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız. Ekranın fontunu belirlemek için `Spi_Glcd_Set_Font` yordamını kullanınız. Eğer herhangi bir font seçilmemiş ise, kütüphane tarafından sağlanan 5x8 font kullanılır.
Örnek	`Spi_Glcd_Write_Text('Merhaba!', 0, 0, 1);`

Spi_Glcd_Image

Yapısı	`void Spi_Glcd_Image(const char * image);`
Tanımı	Bit-Resim-Dosyası (bitmap) görüntüsünü GLCD üzerinde gösterir. `image` parametresi 1024 byte'lık bir dizi gibi biçimlendirilmelidir. mikroC kendi içerisinde bitmap'i LCD'ye çeviren Bitmap-to-LCD editörünü bulundurmaktadır, bu editörü kullanarak bitmap görüntülerinizi LCD üzerinde görüntülenebilecek sabit dizilere çevirebilirsiniz (Derleyici menüsünden; **Tools > Graphic LCD Editor**).
Gereklilikler	GLCD'nin başlangıç durumuna getirilmesi gerekmektedir. `Spi_Glcd_Init`'e bakınız.
Örnek	`Spi_Glcd_Image(benim_resmim);`

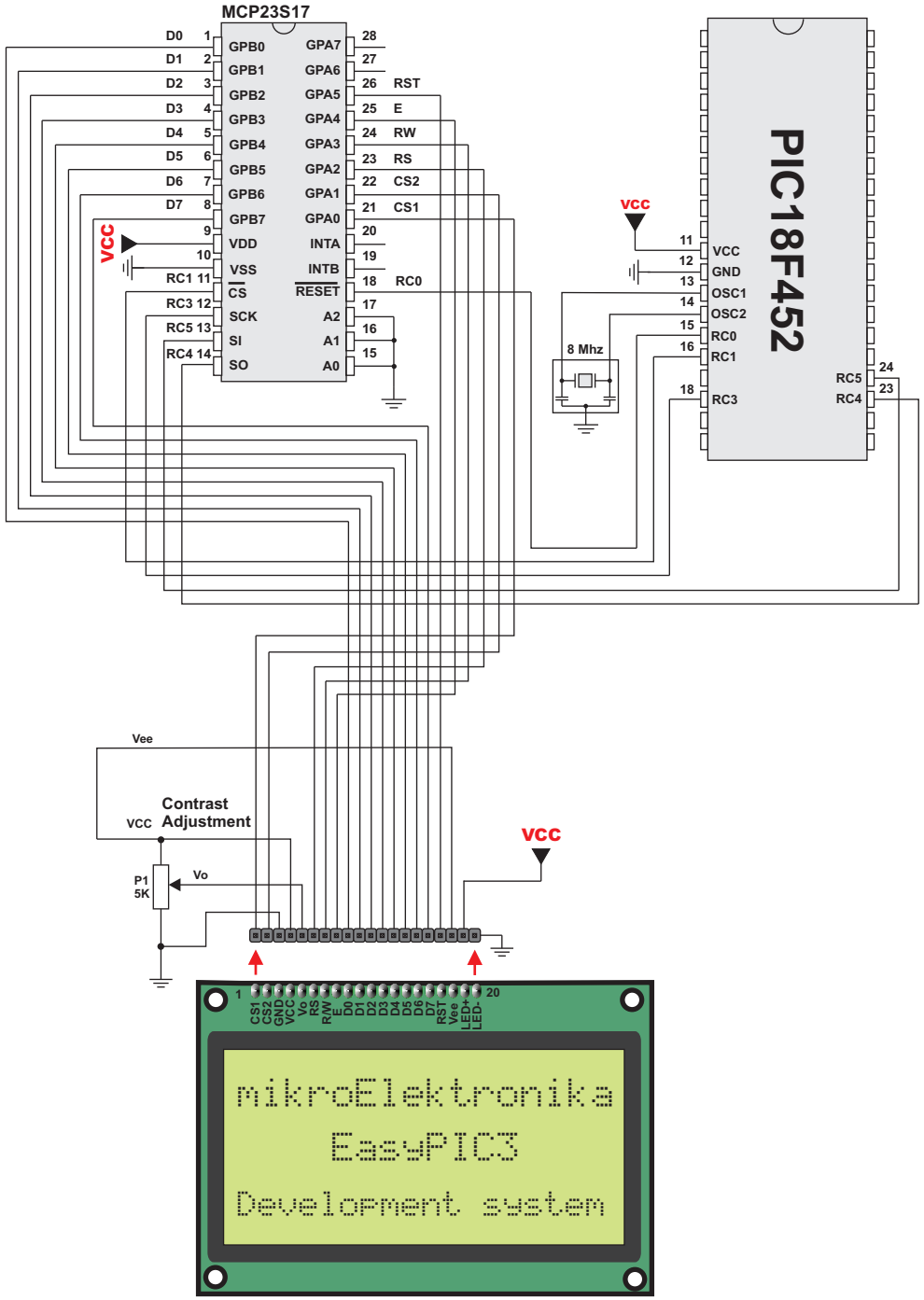
Kütüphane Örneği

Aşağıdaki örnek, seriden paralele dönüştürücü olan MCP23S17'yi kullanarak, SPI yoluyla KS0108 GLCD ile nasıl haberleşileceğini göstermektedir.

```
extern const unsigned short
  truck_bmp[];

char ii;
unsigned int jj;
char *someText;

void delay2S() {
  Delay_ms(2000);
}

void main() {
  ADCON1 |= 0x0F;

  Spi_Init();     // SPI'i baslangic durumuna getir

  Spi_Glcd_Init(0,&PORTC, 0, &PORTC, 1);
  Spi_Glcd_Fill(0xAA);
  delay2S();
  while(1) {
       Spi_Glcd_Fill(0x00);
       Spi_Glcd_Image( truck_bmp );
       delay2S();

       for(jj = 1; jj <= 40; jj++)
         Spi_Glcd_Dot(jj,jj,1);
       delay2S();

       Spi_Glcd_Fill(0x00);
       Spi_Glcd_Line(120, 1, 5,60, 1);
       delay2S();
       Spi_Glcd_Line(12, 42, 5,60, 1);
       delay2S();

       Spi_Glcd_Rectangle(12, 20, 93,57, 1);
       delay2S();




       //devam ediyor..
```

```
        //devam...

        Spi_Glcd_Line(120, 12, 12,60, 1);
        delay2S();

        Spi_Glcd_H_Line(5, 40, 6, 1);
        delay2S();
        Spi_Glcd_Line(0, 12, 120, 60, 1);
        Spi_Glcd_V_Line(7, 63, 127, 1);
        delay2S();

        for(ii = 1; ii <= 10; ii++)
          Spi_Glcd_Circle(63, 32, 3*ii, 1);

        delay2S();
        Spi_Glcd_Box(12, 20, 70, 57, 2);
        delay2S();

        Spi_Glcd_Fill(0x00);

        Spi_Glcd_Set_Font(System3x6, 3, 6, 32);
        someText = "SMALL FONT: 3X6";
        Spi_Glcd_Write_Text(someText, 20, 5, 1);

        Spi_Glcd_Set_Font(FontSystem5x8, 5, 8, 32);
        someText = "Large Font 5x8";
        Spi_Glcd_Write_Text(someText, 3, 4, 1);
        delay2S();
      }
}
```

Donanım Bağlantısı



PORT GENİŞLETİCİ KÜTÜPHANESİ

SPI Port Genişletici Kütüphanesi Microchip'in SPI port genişleticisi olan MCP23S17 ile çalışmayı sağlar. Entegre sayfa 387'deki şemada gösterilen şekilde PIC'e bağlanır.
Not: PIC'in donanımında SPI biriminin olması gereklidir.
Not: Port genişletici başlangıç durumuna getirilmeden önce `Spi_Init` çağrılmalıdır.

Kütüphane Yordamları

```
Expander_Init
PortExpanderSelect
PortExpanderUnSelect
Expander_Read_Byte
Expander_Write_Byte
Expander_Set_Mode
Expander_Read_Array
Expander_Write_Array
Expander_Read_PortA
Expander_Read_PortB
Expander_Read_ArrayPortA
Expander_Read_ArrayPortB
Expander_Write_PortA
Expander_Write_PortB
Expander_Set_DirectionPortA
Expander_Set_DirectionPortB
Expander_Set_PullUpsPortA
Expander_Set_PullUpsPortB
```

Expander_Init

Yapısı	`void Expander_Init(char ModuleAddress, unsigned int * rstport, unsigned int rstpin, unsigned int * csport, unsigned int cspin);`
Tanımı	SPI haberleşmeyi kurar ve genişleticiyi başlangıç durumuna getirir. `RstPort` ve `RstPin`, hangi portun hangi pininin SPI genişleticinin reset pinine bağlanacağını gösterir. `CSPort` ve `CSPin` de hangi port ve hangi pinin SPI genişleticinin CS pinine bağlanacağını gösterir. `moduleaddress` SPI genişleticinin adresidir (Spi genişletici üzerindeki A0, A1 ve A2 pinlerinin VCC veya GND'a bağlanması ile oluşturulmuş donanımsal adres.)
Gereklilikler	`Spi_Init`, Port genişletici başlangıç durumuna getirilmeden önce çağırılmalıdır. Bu yordam diğer port genişletici yordamlarının kullanılmasından önce çağırılmalıdır.
Örnek	`Expander_Init(0, &PORTC, 0, &PORTC, 1);`

PortExpanderSelect

Yapısı	`void PortExpanderSelect;`
Tanımı	Geçerli port genişleticiyi seçer.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`PortExpanderSelect;`

PortExpanderUnSelect

Yapısı	`void PortExpanderUnSelect;`
Tanımı	Geçerli port genişleticiyi seçili olmayan duruma getirir.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`PortExpanderUnSelect;`

Expander_Read_Byte

Yapısı	`char Expander_Read_Byte(char ModuleAddress, char RegAddress);`
Dönüş	Port genişleticiden okunan bayt.
Tanımı	Fonksiyon `ModuleAddress` adresindeki ve `RegAddress` portu üzerindeki port genişleticiden bayt okur.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Read_Byte(0,1);`

Expander_Write_Byte

Yapısı	`void Expander_Write_Byte(char ModuleAddress,char RegAddress, char Data);`
Dönüş	Hiçbir şey
Tanımı	Bu yordam `ModuleAddress` adresindeki ve `RegAddress` portu üzerindeki port genişleticiye `data` verisini yazar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Write_Byte(0,1,$FF);`

Expander_Set_Mode

Yapısı	`void Expander_Set_Mode(char ModuleAddress, char Mode);`
Dönüş	Hiçbir şey
Tanımı	Port genişletici `Mode`'unu `ModuleAddress` adresi üzerinden kurar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Set_Mode(1,0);`

Expander_Read_ArrayPortA

Yapısı	`void Expander_Read_ArrayPortA(char ModuleAddress, char NoBytes, char DestArray[]);`
Dönüş	Hiçbir şey
Tanımı	`ModuleAddress` ve PortA üzerindeki port genişleticisinden bayt dizilerini (`DestArray`) okur. `NoBytes` okunan baytların sayısını göstermektedir.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Read_PortA(0,1,data);`

Expander_Read_Array

Yapısı	`void Expander_Read_Array(char ModuleAddress, char StartAddress, char NoBytes, char *DestArray);`
Dönüş	Hiçbir şey.
Tanımı	`ModuleAddress` ve `StartAddress` üzerindeki port genişleticisinden bayt dizisini (`DestArray`) okur. `NoBytes` okunan baytların sayısını gösterir.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Read_Array(1,1,5,data);`

Expander_Write_Array

Yapısı	`void Expander_Write_Array(char ModuleAddress, char StartAddress, char NoBytes, char *SourceArray);`
Dönüş	Hiçbir şey
Tanımı	`ModuleAddress` ve `StartAddress` üzerindeki port genişleticisine `SourceArray` bayt dizisini yazar. `NoBytes` yazılan baytların sayısını gösterir.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Write_Array(1,1,5,data);`

Expander_Read_PortA

Yapısı	`char Expander_Read_PortA(char Address);`
Dönüş	Okunan bayt
Tanımı	`ModuleAddress` ve PortA üzerindeki port genişleticisinden bir bayt okur.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Read_PortA(1);`

Expander_Read_ArrayPortB

Yapısı	`void Expander_Read_ArrayPortB(char ModuleAddress, char NoBytes, char DestArray[]);`
Dönüş	Hiçbir şey
Tanımı	`ModuleAddress` ve PortB üzerindeki port genişleticisinden bayt dizilerini (`DestArray`) okur. `NoBytes` okunan baytların sayısını göstermektedir.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Read_PortB(0,8,data);`

Expander_Write_PortA

Yapısı	`void Expander_Write_PortA(char ModuleAddress, char Data);`
Dönüş	Hiçbir şey
Tanımı	`ModuleAddress` ve PortA üzerindeki port genişleticiye bayt (`data`) yazar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_write_PortA(3,$FF);`

Expander_Write_PortB

Yapısı	`void Expander_Write_PortB(char ModuleAddress, char Data);`
Dönüş	Hiçbir şey
Tanımı	`ModuleAddress` ve PortB üzerindeki port genişleticisine bayt (`data`) yazar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_write_PortB(2,$FF);`

Expander_Set_DirectionPortA

Yapısı	`void Expander_Set_DirectionPortA(char ModuleAddress, char Data);`
Tanımı	Port genişleticinin PortA pinlerini çıkış veya giriş olarak ayarlar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Set_DirectionPortA(0,$FF);`

Expander_Set_DirectionPortB

Yapısı	`void Expander_Set_DirectionPortB(char ModuleAddress, char Data);`
Tanımı	Port genişleticinin PortB pinlerini çıkış veya giriş olarak ayarlar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Set_DirectionPortB(0,$FF);`

Expander_Set_PullUpsPortA

Yapısı	`void Expander_Set_PullUpsPortA(char ModuleAddress, char Data);`
Tanımı	Port genişleticinin PortA pinlerini pullup veya pulldown olarak ayarlar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Set_PullUpsPortA(0,$FF);`

Expander_Set_PullUpsPortB

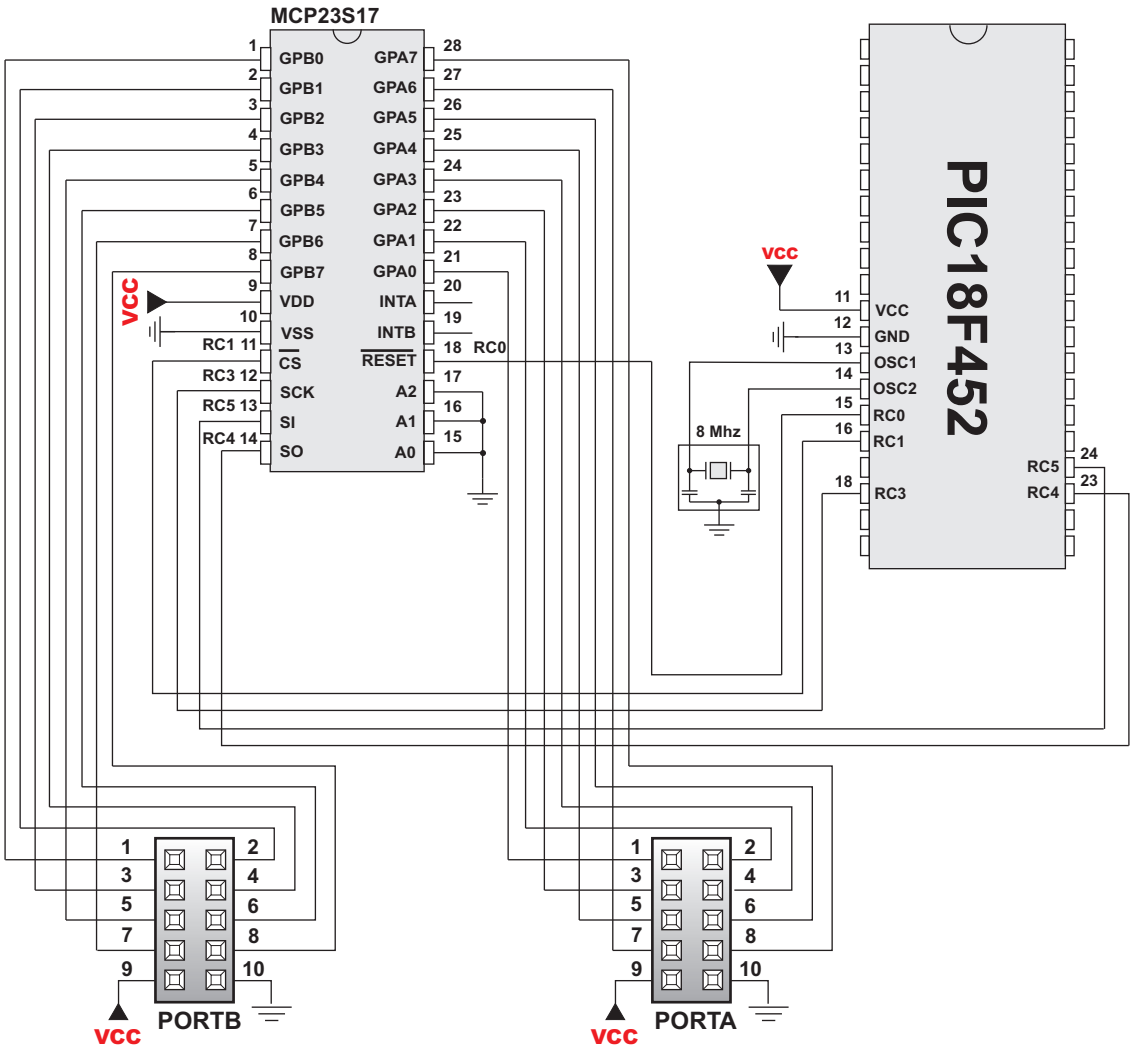
Yapısı	`void Expander_Set_PullUpsPortB(char ModuleAddress, char Data);`
Tanımı	Port genişleticinin PortB pinlerini pullup veya pulldown olarak ayarlar.
Gereklilikler	Port genişletici başlangıç durumuna getirilmiş olmalıdır. `Expander_Init`'e bakınız.
Örnek	`Expander_Set_PullUpsPortB(0,$FF);`

Kütüphane Örneği

Örnek, port genişletici olan MCP23S17 ile nasıl haberleşileceğini göstermektedir.

```
unsigned char  i;

void main(){
  ADCON1 |= 0x0f;
  TRISB = 0x00;
  LATB  = 0xFF;
  Delay_ms(2000);

  Spi_Init();     // SPI modulunu baslangic durumuna getir

  Expander_Init(0, &PORTC, 0, &PORTC, 1); /* Port genisleticiyi baslangic
                                             durumuna getir */

  Expander_Set_DirectionPortA(0, 0);    /* Genisleticinin porta'sini
                                           cikis olarak ayarla */

  Expander_Set_DirectionPortB(0,0xFF);  /* Genisleticinin portb'sini
                                           giris olarak ayarla */
  Expander_Set_PullUpsPortB(0,0xFF);    /* Genisleticinin portb'sinin
                                           tum pinlerini Pullup'a kur */

  i = 0;
  while(1) {

      // Genisleticinin porta'sina i'yi yaz
      Expander_Write_PortA(0, i++);

      // Genisleticinin portb'sini oku ve onu PIC'in LATB'sine yaz
      LATB = Expander_Read_PortB(0);

      Delay_ms(20);
  }
}
```

Donanım Bağlantısı



SPI LCD Kütüphanesi (4-bit arabirimli)

mikroC genel kullanımlı ve 4-bit arabirimli LCD'ler ile SPI arabirimi üzerinden haberleşme için bir kütüphane sağlar. PIC'in ve SPI LCD'nin donanım bağlantısı bölüm sonunda gösterilmiştir.
Not: SPI LCD başlangıç durumuna getirilmeden önce `Spi_Init`' in çağırılması gerekmektedir.

Kütüphane Yordamları

```
Spi_Lcd_Config
Spi_Lcd_Init
Spi_Lcd_Out
Spi_Lcd_Out_Cp
Spi_Lcd_Chr
Spi_Lcd_Chr_Cp
Spi_Lcd_Cmd
```

Spi_Lcd_Config

Yapısı	`void Spi_Lcd_Config(char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin);`
Tanımı	Belirttiğiniz pin ayarları ile (reset pini ve chip select pini) ve SPI arayüzü vasıtasıyla LCD'yi başlangıç durumuna getirir.
Gereklilikler	SPI LCD başlatılmadan önce `Spi_Init` çağırılmalıdır.
Örnek	`Spi_Lcd_Config(0, &PORTB, 1, &PORTB, 0);`

Spi_Lcd_Init

Yapısı	`void Spi_Lcd_Init();`
Tanımı	Geçerli pin ayarları ile LCD'yi başlangıç durumuna getirir (bölüm sonundaki bağlantı şemasına bakınız).
Gereklilikler	SPI LCD başlangıç durumuna getirilmeden önce `Spi_Init` çağırılmalıdır.
Örnek	`Spi_Lcd_Init();`

Spi_Lcd_Out

Yapısı	`void Spi_Lcd_Out(char row, char column, char *text);`
Tanımı	Metni LCD'nin belirtilen satır ve kolonuna (`row` ve `col` parametreleri) yazar. Sabit karakter dizileri (literaller) ve değişken karakter dizileri bir metin (`text`) olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd_Config` veya `Spi_Lcd_Init`'e bakınız.
Örnek	`Spi_Lcd_Out(1, 3, "Merhaba!");`

Spi_Lcd_Out_Cp

Yapısı	`void Spi_Lcd_Out_CP(char *text);`
Tanımı	Metni (`text`) LCD'ye imleç pozisyonundan başlayarak yazar. Sabit karakter dizileri (literaller) ve değişken karakter dizileri bir metin (`text`) olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd_Config` veya `Spi_Lcd_Init`'e bakınız.
Örnek	`Spi_Lcd_Out_Cp("Merhaba!"); /* "merhaba!" yazisini gecerli imlec pozisyonunda yaz. */`

Spi_Lcd_Chr

Yapısı	`void Spi_Lcd_Chr(char Row, char Column, char Out_Char);`
Tanımı	Karakteri (`Out_Char`) LCD'nin belirtilen satır ve kolonuna (`row` ve `col`) yazar. Sabitler (literaller) ve değişkenler, `Out_Char` olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd_Config` veya `Spi_Lcd_Init`'e bakınız.
Örnek	`Spi_Lcd_Chr(2, 3, "i");`

Spi_Lcd_Chr_Cp

Yapısı	`void Spi_Lcd_Chr_Cp(char Out_Char);`
Tanımı	Karakteri (`Out_Char`) LCD üzerinde geçerli imleç pozisyonuna yazar. Sabitler (literaller) ve değişkenler, `Out_Char` olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd_Config` veya `Spi_Lcd_Init`'e bakınız.
Örnek	`Spi_Lcd_Chr_Cp("e");  // "e" yi imlecin bulundugu yere yaz`

Spi_Lcd_Cmd

Yapısı	`void Spi_Lcd_Cmd(char out_char);`
Tanımı	LCD'ye komut (`out_char` parametresi) yollar. Fonksiyona önceden belirtilmiş sabitlerden birini geçebilirsiniz. Tüm geçerli komutlar bir sonraki sayfada verilmiştir.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd_Config` veya `Spi_Lcd_Init`'e bakınız.
Örnek	`Spi_Lcd_Cmd(LCD_CLEAR);  // LCD ekrani temizle`

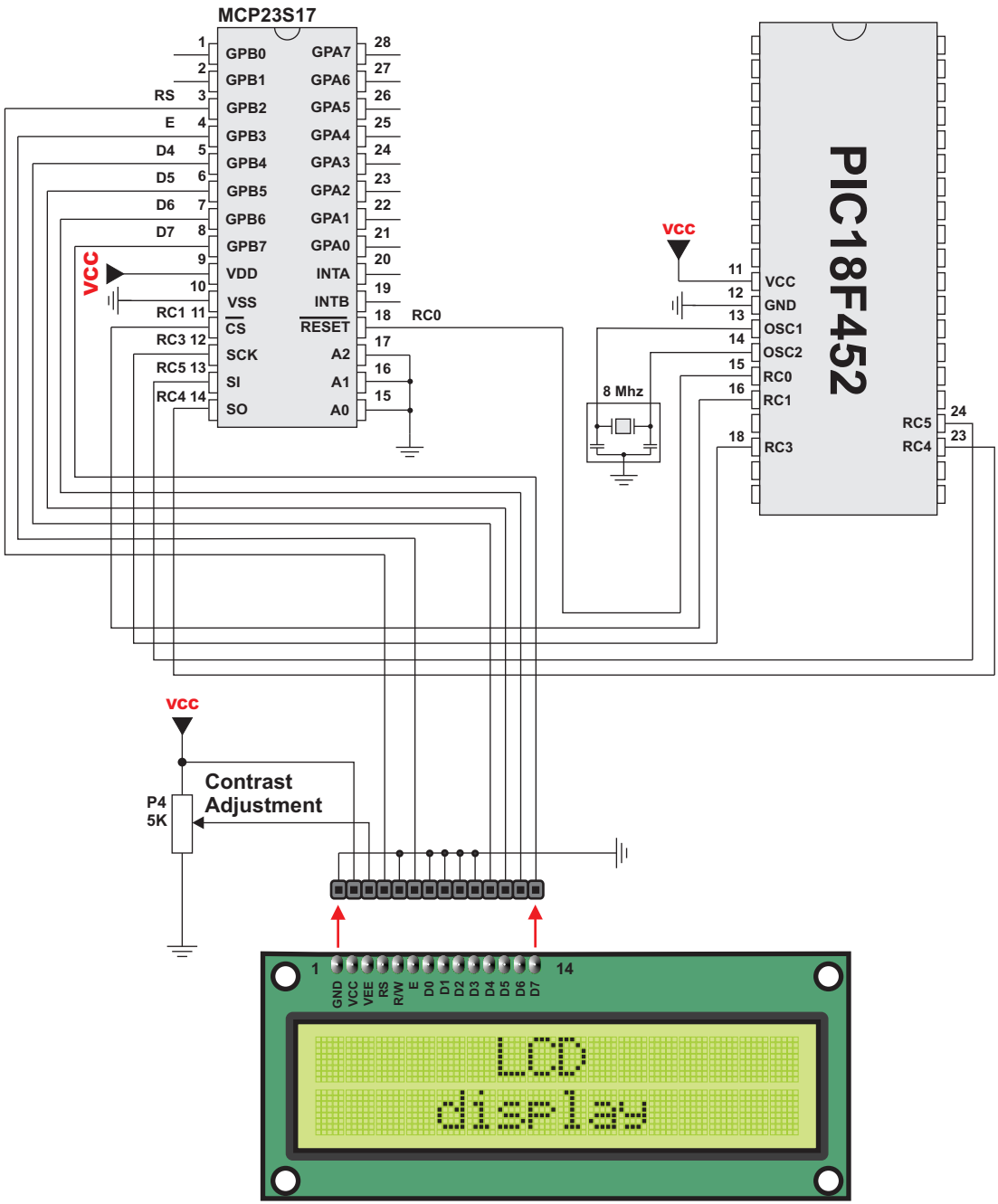
LCD Komutları

LCD Komutu	Amacı
LCD_FIRST_ROW	İmleci birinci satıra taşır
LCD_SECOND_ROW	İmleci ikinci satıra taşır
LCD_THIRD_ROW	İmleci üçüncü satıra taşır
LCD_FOURTH_ROW	İmleci dördüncü satıra taşır
LCD_CLEAR	Ekranı temizler
LCD_RETURN_HOME	İmleci başlangıç pozisyonuna getirir, kaydırılmış bir görüntüyü eski durumuna getirir. Görüntü veri RAM'ının içeriği değişmez.
LCD_CURSOR_OFF	İmleci kapatır
LCD_UNDERLINE_ON	İmlecin altını çizme işlemini açar
LCD_BLINK_CURSOR_ON	İmlecin yanıp-sönme (Blink) işlemini açar
LCD_MOVE_CURSOR_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sola taşır
LCD_MOVE_CURSOR_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sağa taşır
LCD_TURN_ON	LCD ekranını açar
LCD_TURN_OFF	LCD ekranını kapatır
LCD_SHIFT_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sola kaydırır
LCD_SHIFT_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sağa kaydırır

Kütüphane Örneği (Geçerli pin ayarları için)

```
char *text = "mikroElektronika";

void main() {
  Spi_Init();                   // Spi'i baslangic durumuna getir
  Spi_Lcd_Init();               // LCD'yi SPI arayuzuyle baslangic durumuna getir
  Spi_Lcd_Cmd(LCD_CLEAR);       // Ekrani temizle
  Spi_Lcd_Cmd(LCD_CURSOR_OFF);  // Imleci sondur
  Spi_Lcd_Out(1,6, "mikroE");   // LCD'ye metni yaz, 1. satir, 6. kolon'a
  Spi_Lcd_Chr_CP('!');          // ! isaretini de ekle
  Spi_Lcd_Out(2,0, text);       // LCD'ye metni yaz, 2. satır, 0. kolon
  Spi_Lcd_Out(3,1,"mikroE");    // Ikiden fazla satıri olan LCD ler icin ek yazi
  Spi_Lcd_Out(4,15,"mikroE");   // Ikiden fazla satıri olan LCD ler icin ek yazi
} //~!
```

Donanım Bağlantısı



SPI LCD8 Kütüphanesi (8-bit arabirimli)

mikroC genel kullanımlı ve 8-bit arabirimli (Hitachi HD44780 denetleyicili) LCD'ler ile SPI arayüzü üzerinden haberleşme için bir kütüphane sağlar. PIC'in ve SPI LCD'nin donanım bağlantısı bölüm sonunda gösterilmiştir.

Not: SPI LCD8 başlangıç durumuna getirilmeden önce `Spi_Init'` in çağırılması gerekmektedir.

Kütüphane Yordamları

```
Spi_Lcd8_Config
Spi_Lcd8_Init
Spi_Lcd8_Out
Spi_Lcd8_Out_Cp
Spi_Lcd8_Chr
Spi_Lcd8_Chr_Cp
Spi_Lcd8_Cmd
```

Spi_Lcd8_Config

Yapısı	`void Spi_Lcd8_Config(char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin);`
Tanımı	Belirttiğiniz pin ayarları ile (reset pini ve chip select pini) ve SPI arayüzü vasıtasıyla LCD'yi başlangıç durumuna getirir.
Gereklilikler	SPI LCD8 başlangıç durumuna getirilmeden önce `Spi_Init` çağırılmadır.
Örnek	`Spi_Lcd8_Config(0, &PORTB, 1, &PORTB, 0);`

Spi_Lcd8_Init

Yapısı	`void Spi_Lcd8_Init();`
Tanımı	LCD'yi geçerli pin ayarları ile kontrol portu (`ctrlport`) ve veri portu (`dataport`) üzerinden başlangıç durumuna getirir (bölüm sonundaki örneğin şemasına bakın).
Gereklilikler	SPI LCD8 başlangıç durumuna getirilmeden önce `Spi_Init` çağrılmadır.
Örnek	`Spi_Lcd8_Init();`

Spi_Lcd8_Out

Yapısı	`void Spi_Lcd8_Out(unsigned short row, unsigned short column, char *text);`
Tanımı	Metni LCD'nin belirtilen satır ve kolonuna (`row` ve `column` parametreleri) yazar. Sabit karakter dizileri (literaller) ve değişken karakter dizileri bir metin (`text`) olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd8_Config` veya `Spi_Lcd8_Init`'e bakınız
Örnek	`Spi_Lcd8_Out(1, 3, "Merhaba!"); /* 1. satir 3. karakterden basla ve "Merhaba!" yaz */`

Spi_Lcd8_Out_Cp

Yapısı	`void Spi_Lcd8_Out_CP(char *text);`
Tanımı	Metni (`text`) LCD'ye imleç pozisyonundan başlayarak yazar. Sabit karakter dizileri (literaller) ve değişken karakter dizileri bir metin (`text`) olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd8_Config` veya `Spi_Lcd8_Init`'e bakınız
Örnek	`Spi_Lcd8_Out_Cp("merhaba!"); /* "merhaba!" yazisini gecerli imlec pozisyonunda yaz. */`

Spi_Lcd8_Chr

Yapısı	`void Spi_Lcd8_Chr(unsigned short row, unsigned short column, char out_char);`
Tanımı	Karakteri (`out_char`) LCD'nin belirtilen satır ve kolonuna (`row` ve `column`) yazar. Sabitler (literaller) ve değişkenler, `out_char` olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd8_Config` veya `Spi_Lcd8_Init`'e bakınız
Örnek	`Spi_Lcd8_Out(2, 3, 'i'); // 2. satır 3. sutuna 'i' yaz`

Spi_Lcd8_Chr_Cp

Yapısı	`void Spi_Lcd8_Chr_CP(char out_char);`
Tanımı	Karakteri (`out_char`) LCD üzerinde geçerli imleç pozisyonuna yazar. Sabitler (literaller) ve değişkenler, `out_char` olarak geçebilirler.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd8_Config` veya `Spi_Lcd8_Init`'e bakınız
Örnek	`Spi_Lcd8_Chr_Cp('e'); // 'e' yi imlecin bulundugu yere yaz`

Spi_Lcd8_Cmd

Yapısı	`void Spi_Lcd8_Cmd(char out_char);`
Tanımı	LCD'ye komut (`command`) yollar. Fonksiyona önceden belirtilmiş sabitlerden birini geçebilirsiniz. Tüm geçerli komutlar bir sonraki sayfada verilmiştir.
Gereklilikler	LCD'li port başlangıç durumuna getirilmiş olmalıdır. `Spi_Lcd8_Config` veya `Spi_Lcd8_Init`'e bakınız
Örnek	`Spi_Lcd8_Cmd(LCD_CLEAR); // LCD ekrani temizle`

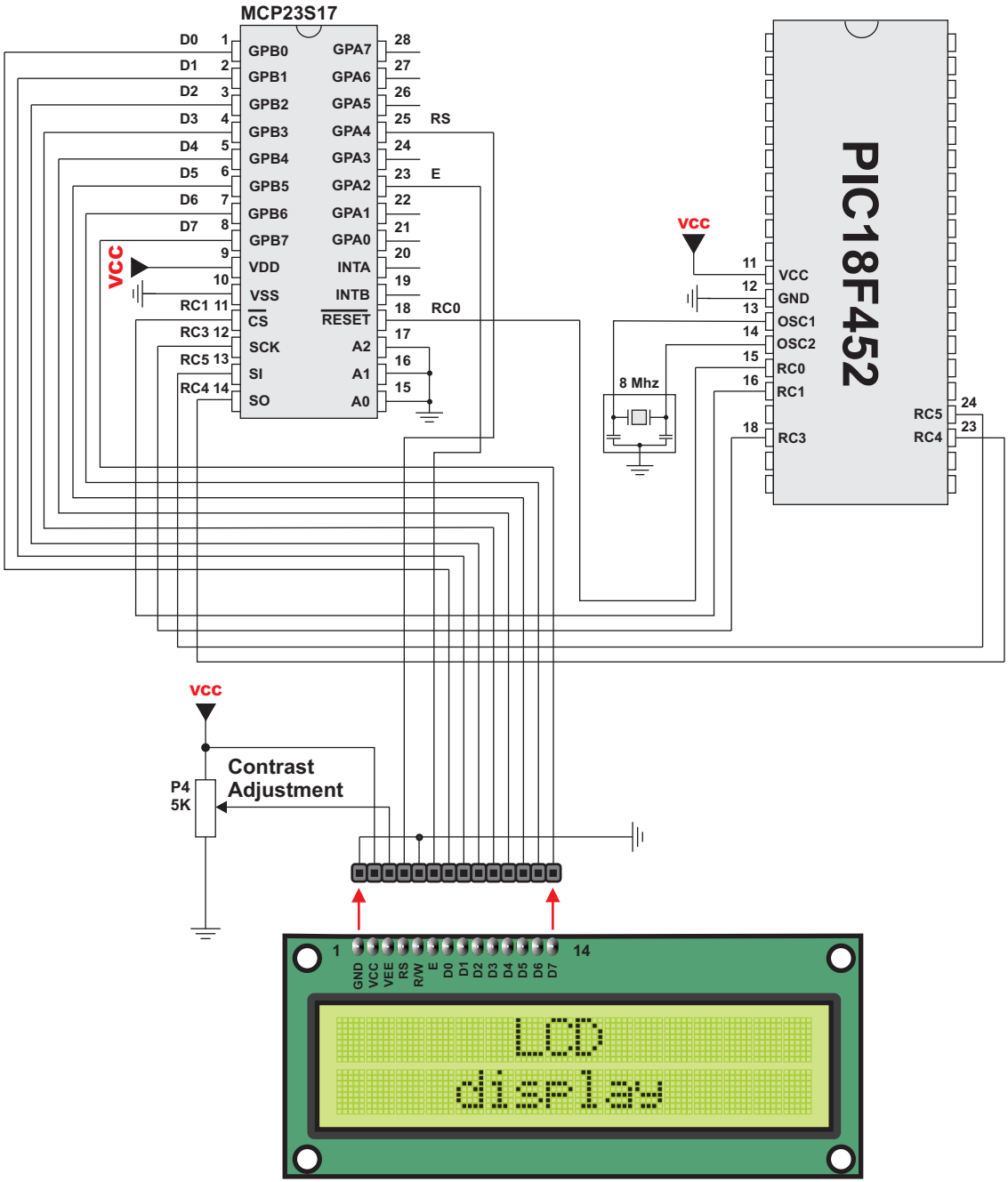
LCD Komutları

LCD Komutu	Amacı
LCD_FIRST_ROW	İmleci birinci satıra taşır
LCD_SECOND_ROW	İmleci ikinci satıra taşır
LCD_THIRD_ROW	İmleci üçüncü satıra taşır
LCD_FOURTH_ROW	İmleci dördüncü satıra taşır
LCD_CLEAR	Ekranı temizler
LCD_RETURN_HOME	İmleci başlangıç pozisyonuna getirir, kaydırılmış bir görüntüyü eski durumuna getirir. Görüntü veri RAM'ının içeriği değişmez.
LCD_CURSOR_OFF	İmleci kapatır
LCD_UNDERLINE_ON	İmlecin altını çizme işlemini açar
LCD_BLINK_CURSOR_ON	İmlecin yanıp-sönme (Blink) işlemini açar
LCD_MOVE_CURSOR_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sola taşır
LCD_MOVE_CURSOR_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden imleci sağa taşır
LCD_TURN_ON	LCD ekranını açar
LCD_TURN_OFF	LCD ekranını kapatır
LCD_SHIFT_LEFT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sola kaydırır
LCD_SHIFT_RIGHT	Görüntü veri RAM'ının içeriğini değiştirmeden veriyi sağa kaydırır

Kütüphane Örneği (Geçerli pin ayarları için)

```
char *text = "mikroE";

void main() {
  Spi_Init();                          // Spi'i baslangic durumuna getir
  Spi_Lcd8_Init();                     // LCD'yi 8 bit modda, SPI arayuzuyle
                                       // baslangic durumuna getir
  Spi_Lcd8_Cmd(LCD_CLEAR);             // Ekrani temizle
  Spi_Lcd8_Cmd(LCD_CURSOR_OFF);        // Imleci sondur (gizle)
  Spi_Lcd8_Out(1,6, text);             // LCD'ye metni yaz, 1. satir 6. kolon'a
  Spi_Lcd8_Chr_CP('!');                // '!' isaretini de ekle
  Spi_Lcd8_Out(2,0, "mikroelektronika");// LCD'ye metni yaz, 2. satir 0. kolon
  Spi_Lcd8_Out(3,1, text);             // 2 den fazla satiri olan lcdler icin ek yazi
  Spi_Lcd8_Out(4,15, text);            // 2 den fazla satiri olan lcdler icin ek yazi
} //~!
```

Donanım Bağlantısı



SPI T6963C Grafik LCD Kütüphanesi

mikroC çeşitli boyutlardaki Toshiba T6963C Grafik LCD'ler üzerinde SPI yoluyla yazma ve çizme için bir kütüphane sağlar.

Not: SPI LCD başlangıç durumuna getirilmeden önce `Spi_Init` çağırılmalıdır.

Kütüphane Yordamları

```
Spi_T6963C_Config
Spi_T6963C_writeData
Spi_T6963C_writeCommand
Spi_T6963C_setPtr
Spi_T6963C_waitReady
Spi_T6963C_fill
Spi_T6963C_dot
Spi_T6963C_write_char
Spi_T6963C_write_text
Spi_T6963C_line
Spi_T6963C_rectangle
Spi_T6963C_box
Spi_T6963C_circle
Spi_T6963C_image
Spi_T6963C_sprite
Spi_T6963C_set_cursor
Spi_T6963C_clearBit
Spi_T6963C_setBit
Spi_T6963C_negBit
Spi_T6963C_displayGrPanel
Spi_T6963C_displayTxtPanel
Spi_T6963C_setGrPanel
Spi_T6963C_setTxtPanel
Spi_T6963C_panelFill
Spi_T6963C_grFill
Spi_T6963C_txtFill
Spi_T6963C_cursor_height
Spi_T6963C_graphics
Spi_T6963C_text
Spi_T6963C_cursor
Spi_T6963C_cursor_blink
Spi_T6963C_Config_240x128
Spi_T6963C_Config_240x64
```

Spi_T6963C_Config

Yapısı	`void Spi_T6963C_Config(unsigned int width, unsigned char height, unsigned char fntW, char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin, unsigned char wr, unsigned char rd, unsigned char cd, unsigned char rst);`
Tanımı	Grafik LCD denetleyicisini başlangıç durumuna getirir. Bu fonksiyon, SPI T6963C kütüphanesinin diğer yordamlarının kullanımından önce çağırılmış olmalıdır. `width` – görüntü birimindeki (x) yatay nokta sayısı. `height` – görüntü birimindeki (y) düşey nokta sayısı. `fntW`– font genişliği, text karakterindeki nokta sayısı donanıma göre belirlenmelidir. `DeviceAddress`- Aygıt Adresi `data` – veri hatlarının bağlandığı port'un adresi. `cntrl` – kontrol hatlarının bağlandığı port'un adresi. `wr` – kontrol portundaki !WR hattı bit numarası `rd` – kontrol portundaki !RD hattı bit numarası `cd` – kontrol portundaki C/D hattı bit numarası `rst` – kontrol portundaki !RST hattı bit numarası Görüntü Ünitesi RAM'i: Kütüphane RAM miktarını bilemez. Kütüphane RAM'i panellere böler; komple bir panel; bir grafik paneli ve onu izleyen bir metin panelinden oluşur. Programcı donanımında kaç panel olduğunu bilmelidir.
Gereklilikler	SPI Toshiba T6963C Grafik LCD başlatılmadan önce `Spi_Init` çağırılmalıdır.

Örnek	`Spi_T6963C_Config(240, 64, 8, 0, &PORTB, 1, &PORTB, 0, 0, 1, 3, 4);`
	`/*`
	`* Ekrani 240 nokta genisliginde ve 64 nokta yuksekliginde`
	`* baslangic durumuna getir`
	`* 8 bit karakter genisligi`
	`* Reset pini PORTB.1 uzerinde`
	`* Entegre secme pini PORTB.0 uzerinde`
	`* 0'inci bit !WR`
	`* 1'inci bit !RD`
	`* 3'uncu bit C/D`
	`* 4'uncu bit RST`
	`* Entegre aktif, ters cevirme aktif, 8x8 font kutuphane fontu`
	`* kullaniliyor`
	`* Entegre adresi 0`
	`*/`

Spi_T6963C_writeData

Yapısı	`void Spi_T6963C_writeData(unsigned char data);`
Tanımı	Yordam SPI T6963C denetleyicisine veri yazar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_writeData(AddrL);`

Spi_T6963C_writeCommand

Yapısı	`void Spi_T6963C_writeCommand(unsigned char data);`
Tanımı	Yordam SPI T6963C denetleyicisine komut yazar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_writeCommand(T6963C_CURSOR_POINTER_SET);`

Spi_T6963C_setPtr

Yapısı	`void Spi_T6963C_setPtr(unsigned int addr, unsigned char t);`
Tanımı	Bu yordam `addr` hafıza işaretçisini `t` komutu için ayarlar.
Gereklilikler	GLCD başlatılmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_setPtr(T6963C_grHomeAddr + start, T6963C_ADDRESS_POINTER_SET);`

Spi_T6963C_waitReady

Yapısı	`void Spi_T6963C_waitReady();`
Tanımı	Bu yordam durum baytına, hazır durumu oluşuncaya kadar döngü içinde bakar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_waitReady();`

Spi_T6963C_fill

Yapısı	`void Spi_T6963C_fill(unsigned char data, unsigned int start, unsigned int len);`
Tanımı	Bu yordam denetleyicinin belleğini istenilen başlangıç adresinden başlayarak istenilen uzunluktaki bölge boyunca verilen baytla doldurur.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_fill(0x33,0x00FF,0x000F);`

Spi_T6963C_dot

Yapısı	`void Spi_T6963C_dot(int x, int y, unsigned char color);`
Tanımı	Bu yordam geçerli grafik çalışma panelinde bir nokta koyar. (`x0,y0`) piksel noktasında renk seçenekleri olarak T6963C_[WHITE[BLACK]].
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_dot(x0, y0, pcolor);`

Spi_T6963C_write_char

Yapısı	`void Spi_T6963C_write_char(unsigned char c, unsigned char x, unsigned char y, unsigned char mode);`
Tanımı	Bu yordam geçerli metin çalışma panelinde bir karakter yazar. Karakter `c`'yi, `x` satırı ve `y` kolonuna yazar. mode = T6963C_ROM_MODE_[OR/EXOR/AND].
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_write_char("A",22,23,AND);`

Spi_T6963C_write_text

Yapısı	`void Spi_T6963C_write_text(unsigned char *str, unsigned char x, unsigned char y, unsigned char mode);`
Tanımı	Bu yordam geçerli metin çalışma paneline bir karakter dizisini yazar. Diziyi `x` satırı ve `y` kolonuna yazar. mode = T6963C_ROM_MODE_[OR/EXOR/AND].
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_write_text("GLCD KUTUPHANE ORNEGI, MERHABA !", 0, 0, T6963C_ROM_MODE_XOR);`

Spi_T6963C_line

Yapısı	`void Spi_T6963C_line(int px0, int py0, int px1, int py1, unsigned char pcolor);`
Tanımı	Bu yordam geçerli grafik çalışma paneline (`x0,y0`)'dan (`x1,y1`)'e bir çizgi çizer. `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_line(0, 0, 239, 127, T6963C_WHITE);`

Spi_T6963C_rectangle

Yapısı	`void Spi_T6963C_rectangle(int x0, int y0, int x1, int y1, unsigned char pcolor);`
Tanımı	Bu yordam geçerli grafik çalışma paneline (`x0,y0`) ve (`x1,y1`) ile verilen dikdörtgen kenarlarını çizer. `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE);`

Spi_T6963C_box

Yapısı	`void Spi_T6963C_box(int x0, int y0, int x1, int y1, unsigned char pcolor);`
Tanımı	Bu yordam geçerli grafik çalışma paneline (`x0,y0`) ve (`x1,y1`) ile verilen dikdörtgen içine dolu bir kutu çizer. `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_box(0, 119, 239, 127, T6963C_WHITE);`

Spi_T6963C_circle

Yapısı	`void Spi_T6963C_circle(int x, int y, long r, unsigned char pcolor);`
Tanımı	Bu yordam geçerli grafik çalışma paneline merkezi (`x,y`) ve yarı-çapı `r` olan bir daire çizer. `pcolor` = T6963C_[WHITE[BLACK]]
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_circle(120, 64, 110, T6963C_WHITE);`

Spi_T6963C_image

Yapısı	`void Spi_T6963C_image(const char *pic);`
Tanımı	Bu yordam geçerli grafik çalışma panelinde grafik alanını MCU işaretçisini kullanarak doldurur. İşaretçinin gösterdiği veri, ekranın geometrisi ile uygun olmalıdır; örneğin 240x128 bir görüntü birimi için ilgili veri (240/8)*128 = 3840 baytlık bir dizin olmalıdır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_image(benim_resmim);`

Spi_T6963C_sprite

Yapısı	`void Spi_T6963C_sprite(unsigned char px, unsigned char py, const char *pic, unsigned char sx, unsigned char sy);`
Tanımı	Bu yordam geçerli grafik çalışma panelinde `(px,py)` ve `(px+sx,py+sy)` ile verilen dikdörtgen'in alanını MCU işaretçisinin gösterdiği resim ile doldurur. `sx` ve `sy` resmin boyutunda olmalıdır. MCU `sx* sy` byte'lık bir dizin olmalıdır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_sprite(76, 4, einstein, 88, 119); //bir resimcik ciz`

Spi_T6963C_set_cursor

Yapısı	`void Spi_T6963C_set_cursor(unsigned char x, unsigned char y);`
Tanımı	Bu yordam imleçi (`x`) satır ve (`y`) kolonuna getirir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_set_cursor(cposx, cposy);`

Spi_T6963C_clearBit

Yapısı	`void Spi_T6963C_clearBit(char b);`
Tanımı	Kontrol bitini sıfırlar
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_clearBit(b);`

Spi_T6963C_setBit

Yapısı	`void Spi_T6963C_setBit(char b);`
Tanımı	Kontrol bitini bir yapar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_setBit(b);`

Spi_T6963C_negBit

Yapısı	`void Spi_T6963C_negBit(char b);`
Tanımı	Kontrol bitini evriğine (tersine) döndürür.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_negBit(b);`

Spi_T6963C_displayGrPanel

Yapısı	`void Spi_T6963C_waitReady(unsigned int n);`
Tanımı	Numarası `(n)` olan Grafik Panel'ini gösterir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_displayGrPanel(n);`

Spi_T6963C_displayTxtPanel

Yapısı	`void Spi_T6963C_displayTxtPanel(unsigned int n);`
Tanımı	Numarası `(n)` olan Metin Panel'ini gösterir.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_displayTxtPanel(n);`

Spi_T6963C_setGrPanel

Yapısı	`void Spi_T6963C_setGrPanel(unsigned int n);`
Tanımı	Panel numarası `(n)` olan Grafik Paneli için grafik başlangıç adresini hesaplar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_setGrPanel(n);`

Spi_T6963C_setTxtPanel

Yapısı	`void Spi_T6963C_setTxtPanel(unsigned int n);`
Tanımı	Panel numarası `(n)` olan Metin Paneli için metin başlangıç adresini hesaplar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_setTxtPanel(n);`

Spi_T6963C_panelFill

Yapısı	`void Spi_T6963C_panelFill(unsigned int v);`
Tanımı	Tüm #n panelini `v` bitmapi (görüntünün tüm bitlerini içeren resim) ile doldurur. (Ekranı temizlemek için parametre olarak 0 kullanarak çağırınız).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_panelFill(v);`

Spi_T6963C_grFill

Yapısı	`void Spi_T6963C_grFill(unsigned int v);`
Tanımı	Grafik #n panelini `v` bitmapi (görüntünün tüm bitlerini içeren resim) ile doldurur. (Ekranı temizlemek için parametre olarak 0 kullanarak çağırınız).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_grFill(v);`

Spi_T6963C_txtFill

Yapısı	`void Spi_T6963C_txtFill(unsigned int v);`
Tanımı	Metin #n panelini `v` + 32 karakteri ile doldurur. (Ekranı temizlemek için parametre olarak 0 kullanarak çağırınız).
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_txtFill(v);`

Spi_T6963C_cursor_height

Yapısı	`void Spi_T6963C_cursor_height(unsigned int n);`
Tanımı	İmleç boyutunu ayarlar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_cursor_height(n);`

Spi_T6963C_graphics

Yapısı	`void Spi_T6963C_graphics(unsigned int n);`
Tanımı	Grafiği açar veya kapatır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_graphics(1);`

Spi_T6963C_text

Yapısı	`void Spi_T6963C_text(unsigned int n);`
Tanımı	Metinleri açar veya kapatır.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_text(1);`

Spi_T6963C_cursor

Yapısı	`void Spi_T6963C_cursor(unsigned int n);`
Tanımı	İmleci gösterir veya gizler.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_cursor(1);`

Spi_T6963C_cursor_blink

Yapısı	`void Spi_T6963C_cursor_blink(unsigned int n);`
Tanımı	İmlecin yanıp-sönme (blinking) özelliğini ayarlar.
Gereklilikler	GLCD başlangıç durumuna getirilmiş olmalıdır. `Spi_T6963C_Config`'e bakınız.
Örnek	`Spi_T6963C_cursor_blink(0);`

Spi_T6963C_Config_240x128

Yapısı	`void Spi_T6963C_Config_240x128();`
Tanımı	T6963 tabanlı GLCD'yi (240x128 pikselli), mikroElektronika GLCD'leri için varsayılan ayarlarla başlatır.
Gereklilikler	SPI Toshiba T6963 grafik LCD başlatılmadan önce `Spi_Init` çağırılmalıdır.
Örnek	`Spi_T6963C_Config_240x128();`

Spi_T6963C_Config_240x64

Yapısı	`void Spi_T6963C_Config_240x64();`
Tanımı	T6963 tabanlı GLCD'yi (240x64 pikselli), mikroElektronika GLCD'leri için varsayılan ayarlarla başlatır.
Gereklilikler	SPI Toshiba T6963 grafik LCD başlatılmadan önce `Spi_Init` çağırılmalıdır.
Örnek	`Spi_T6963C_Config_240x64();`

Kütüphane Örneği

Aşağıdaki program örneği SPI T6963 GLCD'nin gelişmiş yordamlarını test etmektedir.

```
#include         "Spi_T6963C.h"

extern const char mc[] ;
extern const char einstein[] ;

void main(void)
        {
        unsigned char   panel ;        // Kullanilan panel
        unsigned int    i ;            // Genel amacli yazmac
        unsigned char   curs ;         // Imlec gorunurlugu
        unsigned int    cposx, cposy ; // Imlec x-y konumunda
        TRISC = 0 ;                    // portC cikis
        PORTC = 0b00000000 ;           /* Chip Enable
                                          (Yonga etkin),
                                          Tersleme etkin,
                                          8x8 font       */

//devam ediyor...
```

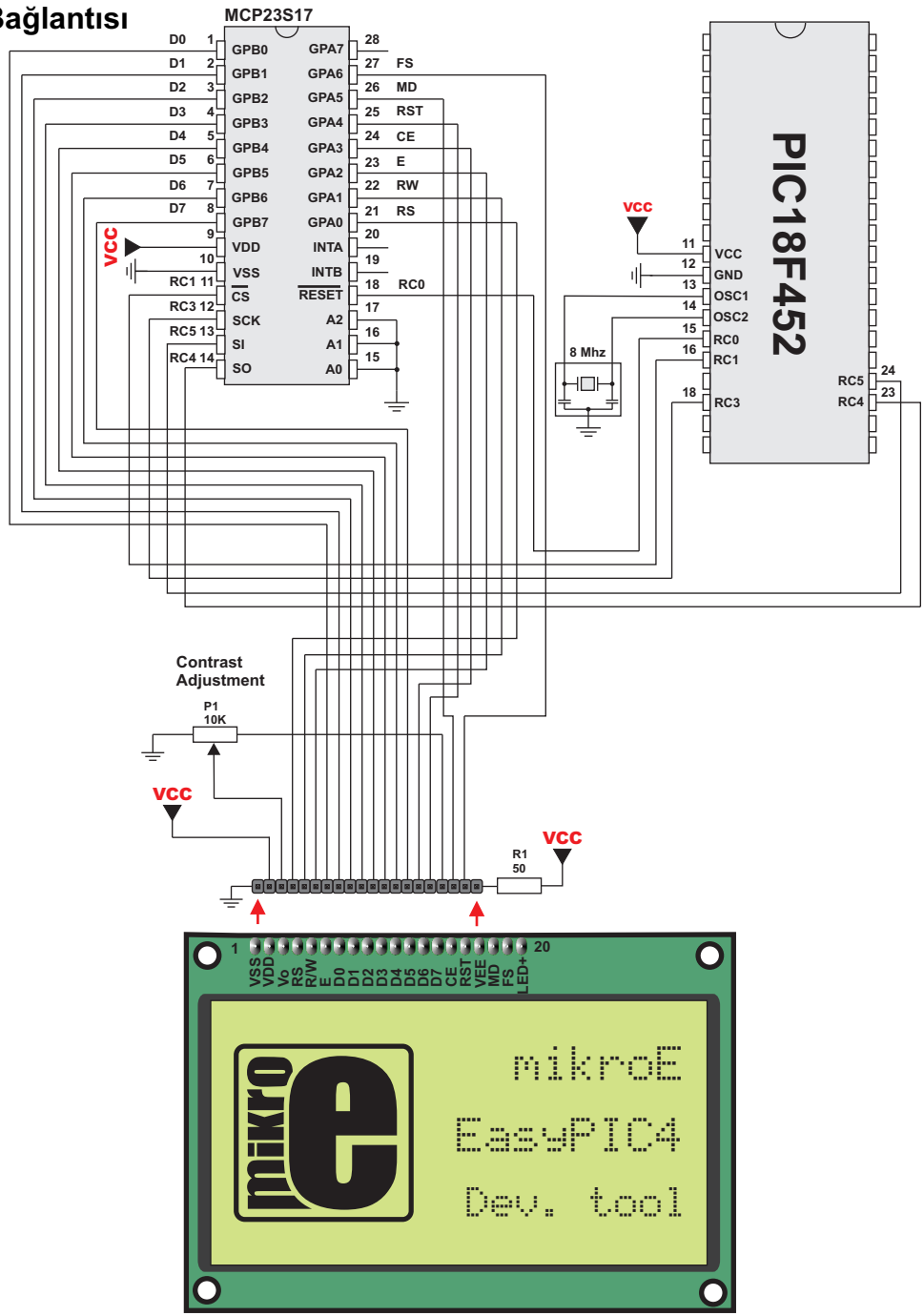
```
//devam...
      /*
         * Goruntu birimini 240 nokta genisliginde ve 128 nokta
         * yuksekliginde baslangic durumuna getir
         * 8 bit karakter genisligi
         * Veri hatti PORTD'de
         * Kontrol hatti PORTC'de
         * 3 No'lu bit !WR
         * 2 No'lu bit !RD
         * 1 No'lu bit C/D
         * 5 No'lu bit RST
      */

        Spi_Init();
        Spi_T6963C_Init_240x128();

        /*
         * Hem grafik hem de metin ekranlarini ayni anda
         * etkinlestir
         */

        Spi_T6963C_graphics(1) ;
        Spi_T6963C_text(1) ;

        panel = 0 ;
        i = 0 ;
        curs = 0 ;
        cposx = cposy = 0 ;

        /*
         * Metin mesaji
         */

        Spi_T6963C_write_text(" GLCD LIBRARY DEMO, WELCOME !", 0, 0,
Spi_T6963C_ROM_MODE_XOR) ;
        Spi_T6963C_write_text(" EINSTEIN WOULD HAVE LIKED mC", 0,
15, Spi_T6963C_ROM_MODE_XOR) ;

        /*
         * Imlec
         */

        Spi_T6963C_cursor_height(8) ;  // 8 piksel yukseklik
        Spi_T6963C_set_cursor(0, 0) ;  // Imleci sol uste tasi
        Spi_T6963C_cursor(0) ;         // Imleci gizle


//devam ediyor...
```

```
//devam...

        /*
         * Dikdortgenler ciz
         */

        Spi_T6963C_rectangle(0, 0, 239, 127, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(20, 20, 219, 107, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(40, 40, 199, 87, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(60, 60, 179, 67, Spi_T6963C_WHITE) ;

        /*
         * Bir carpi isareti çiz
         */

        Spi_T6963C_line(0, 0, 239, 127, Spi_T6963C_WHITE) ;
        Spi_T6963C_line(0, 127, 239, 0, Spi_T6963C_WHITE) ;

        /*
         * Dolu bir kutu ciz
         */

        Spi_T6963C_box(0, 0, 239, 8, Spi_T6963C_WHITE) ;
        Spi_T6963C_box(0, 119, 239, 127, Spi_T6963C_WHITE) ;

        /*
         * Bir cember ciz
         */

        Spi_T6963C_circle(120, 64, 10, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 30, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 50, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 70, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 90, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 110, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 130, Spi_T6963C_WHITE) ;

        // Bir resimcik ciz
        Spi_T6963C_sprite(76, 4, einstein, 88, 119) ;


        Spi_T6963C_setGrPanel(1) ;  // Diger grafik panelini sec

      // Grafik ekranini bir resimle doldur
        Spi_T6963C_image(mc) ;


//devam ediyor...
```

```
//devam...


  for(;;)
          {

// Eger RB1 basilirsa, ekrani grafik panel 0 ve panel 1
// arasinda atlayarak degistir (toggle)

          if(PORTB & 0b00000010)
                  {
                  panel++ ;
                  panel &= 1 ;
                  Spi_T6963C_displayGrPanel(panel) ;
                  Delay_ms(300) ;
                  }

// Eger RB2 basilirsa, sadece grafik panelini goster
          else if(PORTB & 0b00000100)
                  {
                  Spi_T6963C_graphics(1) ;
                  Spi_T6963C_text(0) ;
                  Delay_ms(300) ;
                  }

// Eger RB3 basilirsa, sadece metin panelini goster

          else if(PORTB & 0b00001000)
                  {
                  Spi_T6963C_graphics(0) ;
                  Spi_T6963C_text(1) ;
                  Delay_ms(300) ;
                  }


// Eger RB4 basilirsa grafik ve metin
// panellerini goster

          else if(PORTB & 0b00010000)
                  {
                  Spi_T6963C_graphics(1) ;
                  Spi_T6963C_text(1) ;
                  Delay_ms(300) ;
                  }



//devam ediyor...
```

```
//devam...

                    // Eger RB5 basilirsa, imleci degistir
                  else if(PORTB & 0b00100000)
                          {
                          curs++ ;
                          if(curs == 3) curs = 0 ;
                          switch(curs)
                                  {
                                  case 0:
                                          // imlec yok
                                          Spi_T6963C_cursor(0) ;
                                          break ;
                                  case 1:
                                          // imleci yak/sondur
                                          Spi_T6963C_cursor(1) ;
                                      Spi_T6963C_cursor_blink(1) ;
                                          break ;
                                  case 2:
                                          // imlec sabit
                                          Spi_T6963C_cursor(1) ;

                                      Spi_T6963C_cursor_blink(0) ;
                                          break ;
                                  }
                          Delay_ms(300) ;
                          }

                  /*
                   * Imleci baska yere tasi (gorunur degilse bile)
                   */
                  cposx++ ;
                  if(cposx == Spi_T6963C_txtCols)
                          {
                          cposx = 0 ;
                          cposy++ ;
        if(cposy == Spi_T6963C_grHeight / Spi_T6963C_CHARACTER_HEIGHT)
                                  {
                                  cposy = 0 ;
                                  }
                          }
                  Spi_T6963C_set_cursor(cposx, cposy) ;

                  Delay_ms(100) ;
                  }
          }
```

Donanım Bağlantısı



Toshiba T6963C Graphic LCD (240x128)

Setjmp Kütüphanesi

Bu kütüphane, normal fonksiyon "çağrı ve geri-dönüş" kurallarını pas geçebilmek için fonksiyon ve tip tanımlarını içerir. Bildirimi yapılan tip **jmp_buf** 'tır ve bu özel tip, çağrı (dallanma) ortamını dönüşte tekrar eski haline döndürebilmek için gerekli olan bilgiyi tutmak için uygun bir dizi tipidir.

Tip bildirimi, pic16 ve pic18 mikrodenetleyicileri için sejmp16.h ve setjmp18.h başlık dosyaları içerisinde bulunur. Bu başlık dosyaları derleyicinin "include" dizininde bulunabilir. Bu kütüphanenin gerçeklenmesi pic16 ve pic18 mcu aileleri için farklı yapılmıştır. pic16 ailesi için Setjmp ve Longjmp, setjmp16.h başlık başlık dosyası içerisinde tanımlı makrolar olarak gerçeklenmişlerdir. pic18 ailesi için ise setjmp kütüphane dosyası içerisinde tanımlı fonksiyonlar olarak gerçeklenmişlerdir. pic16 ailesinin yığıt işaretçisini okuma/yazma özelliğinin olmamasından dolayı, Longjmp isteğinden sonraki program çalışması yığıt içeriğine bağlı olarak gerçekleşir. İşte tam da bu nedenle sadece pic16 ailesi için Setjmp ve Longjmp fonksiyonlarının gerçeklenmesi ANSI C standardı ile uyumlu değildir.

Kütüphane Yordamları

```
Setjmp
Longjmp
```

Setjmp

Yapısı	`int setjmp(jmp_buf env);`
Dönüş	Eğer dönüş bu fonksiyonun doğrudan çağrılması sonucunda oluyor ise, 0 döndürür. Eğer dönüş bir longjmp fonksiyonunun çağrılması sonucunda oluyor ise, 0 olmayan bir değer döndürür.
Tanımı	Bu fonksiyon, kendisine ilgili çağrının yapıldığı yeri daha sonra longjmp tarafından kullanılabilmesi için `jmp_buf` içerisinde saklar. jmp_buff dizisi tipinde olan `env` parametresi çağrı ortamının yeniden eski haline gelebilmesi için gerekli olan bilgiyi tutmaya uygundur.
Gereklilikler	Hiçbir şey.
Örnek	`setjmp(buf);`

Longjmp

Yapısı	`void longjmp(jmp_buf env, int val);`
Dönüş	longjmp, setjmp'nin val değeri ile dönmesine neden olur. Eğer val 0 ise 1 döner.
Tanımı	setjmp makrosunun jmp_buf içerisine en son çağrıldığında kaydettiği ortamı tekrar oluşturur. Eğer (setjmp ile) böyle bir çağrı yapılmamış ise veya çağrının yapıldığı fonksiyon sonlanmış ise programın davranışı belirsizdir. jmp_buff dizisi tipinde olan `env` parametresi ilgili setjmp çağrıldığındaki saklanan çağrı ortamını tutar. val parametresi ilgili setjmp'in döndüreceği char değeridir.
Gereklilikler	Longjmp çağrısı, Setjmp çağrısının yapıldığı fonksiyondan dönülmeden önce yapılmalıdır.
Örnek	`longjmp(buf, 2);`

Kütüphane Örneği

Kütüphane örneği, setjmp ve longjmp fonksiyonlarını kullanarak çapraz fonksiyon dallanmasını göstermektedir. Çağırıldığı vakit; Setjmp(), kendi dallanma ortamını daha sonra Longjmp tarafından kullanması için kendi jmp_buf bağımsız değişkeni içerisine kaydeder. Longjmp, diğer taraftan, Setjmp()'in en son çağrılması ile kaydedilmiş ortamı, ilgili jmp_buf bağımsız değişkeni sayesinde yeniden eski hali ile getirir. Bu örneği mikroC'nin Setjmp örnek klasörü içerisinde bulabilirsiniz.

Konunun daha iyi anlaşılması için: Setjmp ve Longjmp fonksiyonları genellikle hata durumlarının kontrolünde kullanılırlar. Öncelikle Setjmp fonksiyonu ile o anki durum kaydedilir. Daha sonra hata durumunun kontrol edilmesi gereken kritik kod çalıştırılır. Herhangi bir hata oluştuğunda Longjmp fonksiyonu çağrılır fakat ilginç şekilde Setjmp fonksiyonundan dönüş yapılır. Üstelik bu dönüş sırasında eski çalışma ortamı da kaydedilen bilgiler ile tekrar kurulur. Böylece başlangıç durumumuza hata durumu oluşmasına rağmen hiçbir bilgi bozulmamış olarak geri dönmüş oluruz.

Zaman (Time) Kütüphanesi

Zaman Kütüphanesi, UNIX zaman biçiminde zaman hesaplamaları için fonksiyonlar ve tip tanımları (definition) içerir. UNIX zaman sisteminde saniyeler bir tarihi andan itibaren şu ana kadar sayılır. Bu tarihi ana "epoch" denir. Bu zaman aralıkları ile çalışılan programlar için çok uygundur. İki UNIX zaman değeri arasındaki aralık, yerel saatin doğruluğu dahilinde, saniye olarak ölçülmüş gerçek bir zaman aralığıdır.

" epoch" nedir?
İlk başta GMT'ye göre (Greenwich Mean Time) 1970 yılının başlangıcı olarak tanımlandı (Julian sistemine göre 01 Ocak 1970). GMT, İngiltere'de yerel saat dilimi için kullanılan geleneksel bir terimdir.

Tip bildirimi, zaman ve tarih kaydına uygun yapılandırılmış bir tip olan TimeStruct' tır. Tip bildirimini, mikroC'nin Zaman Kütüphanesi örnek klasörü içerisinde bulunan timelib.h başlık dosyası içerisinde bulabilirsiniz.

Kütüphane Yordamları

```
Time_dateToEpoch
Time_epochToDate
Time_dateDiff
```

Time_dateToEpoch

Yapısı	`long Time_dateToEpoch(TimeStruct *ts);`
Dönüş	Bu fonksiyon saniyelerin sayısını döndürür.
Tanımı	Bu fonksiyon unix an'ı döndürür (epoch): ts ile işaretlenmiş zaman yapısının 01 Ocak 1970 0h00mn00s 'den itibaren saniye farkını döndürür.
Gereklilikler	Yoktur.
Örnek	`Time_dateToEpoch(&ts1);`

Time_epochToDate

Yapısı	`void Time_epochToDate(long e, TimeStruct *ts);`
Dönüş	Yoktur
Tanımı	Unix'te herhangi bir ana ait olan saniyeleri (`e`'yi)zaman yapısı `ts` içerisine dönüştürecektir (1970'ten itibaren saniye sayısını).
Gereklilikler	hiçbir şey.
Örnek	`Time_epochToDate(epoch, &ts2);`

Time_dateDiff

Yapısı	`long Time_dateDiff(TimeStruct *t1, TimeStruct *t2);`
Dönüş	Bu fonksiyon saniyeler içerisindeki zaman farklılıklarını bir işaretli long olarak döndürür.
Tanımı	Bu fonksiyon iki tarihi birbirleri ile karşılaştırır, saniyeler arasındaki zaman farkını bir işaretli long olarak döndürür. Eğer `t1, t2`'den önde ise sonuç pozitif, eğer ikisi aynı ise sonuç sıfır, eğer `t1, t2`'den geride ise sonuç negatiftir.
Gereklilikler	**Not**: Bu fonksiyon, mikroC Zaman Kütüphanesi Gösterim (Time Library Demo) örnek dizini içerisinde timelib.h başlık dosyasında gerçeklenmiştir.
Örnek	`diff = Time_dateDiff(&ts1, &ts2);`

Kütüphane Örneği

Bu örnek, Zaman Kütüphanesi Örneğini göstermektedir (basitleştirilmiş, C diline yakın - PIC MCU zaman kütüphanesi).

```
#include         "timelib.h"

TimeStruct       ts1, ts2 ;
long     epoch ;
long     diff ;

void main()
         {

         ts1.ss = 0 ;
         ts1.mn = 7 ;
         ts1.hh = 17 ;
         ts1.md = 23 ;
         ts1.mo = 5 ;
         ts1.yy = 2006 ;

         /*
          * ts icerisindeki tarih'in an'i (epoch) nedir?
          */
         epoch = Time_dateToEpoch(&ts1) ;

         /*
          *  1234567890 an'i hangi tarihtir?
          */
         epoch = 1234567890 ;
         Time_epochToDate(epoch, &ts2) ;

         /*
          * Bu iki tarih arasinda kac saniye vardir?
          */
         diff = Time_dateDiff(&ts1, &ts2) ;
         }
```

SÖZLÜK

Access	= erişim	Enumeration	= numaralama
Argument	= bağımsız değişken	Explicit	= açık, belirtilmiş
Array	= dizi	Explorer	= araştırıcı
Assembly	= sembolik makina dili	Exponent	= üs
Assignment	= atama	Expression	= ifade
Bank	= blok	Extension	= uzantı
Baud	= iletilen bit / saniye	Flag	= bayrak
Binary	= ikilik	Floating point	= kayan noktalı
Bitwise	= bit-işlem, bitsel	Fraction	= kesir
Boot	= kök	Function	= işlev, fonksiyon
Border	= sınır, kenar	Global	= global, genel
Branch	= dallanma	Hardware	= donanım
Breakpoint	= durma noktası	Header	= başlık
Built_in	= yerleşik	Hexadecimal	= onaltılık
Bus	= çoklu veri-hattı	Hardware	= donanım
Call	= dallanma	Hint	= ipucu
Chart	= şema	Identifier	= tanıtıcı
Clause	= yan tümce	Implicit	= örtülü, kapalı, içsel
Column	= sütun, kolon	Initialization	= başlangıç durumuna getirme, sıfırlama
Comment	= yorum, açıklama		
Compilation	= derleme	Inline	= kod-içi
Compiler	= derleyici	Integer	= tamsayı
Conditional	= şartlı	Interrupt	= kesme
Consecutive	= ardışık	Invalid	= geçersiz
Constant	= sabit	Iteration	= tekrarlı
Conversion	= dönüşüm	Jump	= atlama
Cursor	= imleç	Keyboard	= klavye
Cycle	= dönüş, devinim	Keypad	= tuştakımı
Data	= veri	Keyword	= anahtar kelime
Debugger	= hata ayıklayıcı	Label	= etiket
Decimal	= onlu	Lexical	= sözlüksel
Declaration	= bildirim	Literal	= sabit
Device	= aygıt, entegre	Linker	= bağlayıcı
Dimension	= boyut	Local	= yerel
Directive	= direktif	Loop	= döngü
Display	= ekran, gösterge	Lvalue	= sol-değer
Edge	= kenar,sınır	Macro	= makro, listecik
Ellipsis	= üç nokta yanyana	MCU	= mikrodenetleyici ünite

Memory = bellek
Microcontroller = mikrodenetleyici
Mod = durum
Module = birim
Namespace = İsim Uzayı
Nested = iç içe geçmiş
Newline = yenisatır
Onewire = tektel
Operand = işlenen
Operation = işlem
Operator = işleç
Parser = ayrıştırıcı, ayraç
Path = yol
Pattern = numune
Pointer = işaretçi
Predefinite = ön tanımlı
Prefix = ön ek
Preprocessor = ön-işlemci
Procedure = prosedür, alt-yordam, alt-program
Programmer = programlayıcı
Punctuator = noktalama işareti
Qualifier = niteleyici
Range = erim, aralık
Reentrancy = yineli-girişlilik
Register = kayıtçı, yazmaç
Return = dönüş
Root directory = kök dizin
Routine = yordam
Row = sıra, dizi
Scope = kapsam
Sector = bölme
Segment = bölme
Sequence = dizi
Simulation = benzeşim
Stack = yığıt
Statement = deyim, komut
Stopwatch = kronometre
String = karakter dizisi, dizgi
Structure = yapı
Subfunction = alt fonksiyon
Subprocedure = alt-prosedür
Subroutine = alt-yordam
Suffix = son ek
Swap = değiş-tokuş
Syntax = söz dizimi
Time stamp = zaman belirteci
Token = dizgecik
Tool = araç
Tool bar = araç cetveli
Typedef = tip tanımı
Union = birlik
Variable = değişken
Visibility = görünürlük
Warning = uyarı
Whitespace = beyaz uzay

KISALTMALAR

ADC = Analog Digital Converter - Analog Sayısal Çevirici
ANSI = American National Standards Intitute - Amerikan Ulusal Standartlar Enstitüsü
ASCII = American Standard Code for Information Interchange - Bilgi Değişimi İçin Amerikan Standart Kodlama Sistemi
ASM = Automatic Storage Management - Otomatik Depolama Yönetimi
CAN = Controller Area Network- Denetleyici Alan Ağı
EEPROM = Electrically Erasable Programmable Read Only Memory - Elektriksel Olarak Silinebilen ve Yazılabilen Salt Okunur Bellek
GLCD = Graphical Liquid Cristal Display - Sıvı Kristal Grafik Göstergesi
GPR = General Purpose Register - Genel Amaçlı Yazmaç
HEX = Hexadecimal - Onaltılı
HID = Human Interface Device - İnsan Arabirim Cihazı
HW = HardWare - Donanım
ICD = In-Circuit Debugger -Devre İçi Hata Ayıklayıcı
IDE = Integrated Development Environment - Tümleşik Geliştirme Ortamı
I2C (IIC) = Inter-Integrated Circuit (Bus) - Tümleşik Devreler Arası (Veri Yolu)
LCD = Liquid Cristal Display - Sıvı Kristal Gösterge
MCU = Micro Controller Unit - Mikro Denetleyici Ünitesi
MMC = Multi Media Card - Çoklu Ortam Kartı
PIC = Peripheral Interface Controller - Çevresel Arabirim Denetleyicisi
PWM = Pulse Width Modulation - Darbe Genişlik Modülasyonu
RAM = Random Access Memory - Rasgele Erişimli Bellek
ROM = Read Only Memory - Salt Okunur Bellek
SDC = Secure Digital Card - Güvenli Sayısal Hafıza Kartı
SFR = Special Function Register - Özel İşlevli Yazmaç
SPI = Serial Peripheral Interface (Bus) - Seri Çevresel Arabirim (Veri Yolu)
UART = Universal Asynchronous Receiver / Transmitter- Evrensel Eşzamansız Alıcı/Verici
USB = Universal Serial Bus - Evrensel Seri Veri Yolu

KAYNAKÇA

mikroC PDF Documents, mikroElektronika Corp., Belgrade, 2004
MikroPASCAL PDF Documents, mikroElektronika Corp., Belgrade, 2004
MikroC PDF Documents, mikroElektronika Corp., Belgrade, 2004
Bilişim Terimleri Karşılıklar Sözlüğü, Türk Bilişim Derneği
Bilim ve Sanat Terimleri Ana Sözlüğü, Türk Dil Kurumu

Bize ulaşın:

Herhangi bir ürünümüz hakkında problem yaşarsanız veya daha fazla bilgi edinmek istiyorsanız, lütfen bize ulaşınız.

Derleyiciler İçin Teknik Destek

mikroC hakkında bir sorunla karşılaşırsanız bizimle bağlantı kurmaktan çekinmeyiniz, bu tür sorunlar karşılıklı görüşmeler ile çözülebilir.

Okul Ve Üniversiteler İçin İndirim

Beti Bilişim Teknolojileri eğitim enstitüleri için özel indirimler sağlar. Eğer mikroC derleyici kitabını sadece eğitim amaçlı alıyorsanız, lütfen bizimle bağlantı kurunuz.

Taşıma Ve Teslimat İle İlgili Problemler

Eğer teslimat gecikmesi veya diğer dağıtıcılarımızla ilgili problem yaşarsanız lütfen bizimle bağlantı kurunuz. Bağlantı için aşağıdaki linki kullanabilirsiniz.

Beti Bilişim Teknolojileri'nin Dağıtıcısı Olmak İster misiniz?

Biz Beti Bilişim Teknolojileri olarak yeni iş arkadaşları ararız. Eğer ürünlerimizin dağıtımında bize yardımcı olmak isterseniz lütfen bize ulaşınız.

Başka

Eğer herhangi bir soru, yorum veya iş teklifiniz varsa, lütfen bizimle bağlantı kurunuz:

Beti Bilişim Teknolojileri Ltd. Şti.
Şerefli Sokak No:40/5 Mebusevleri
ANKARA

Tel: + 90 (312) 222 18 00
Fax:+ 90 (312) 222 18 08
E-mail: info@beti.com.tr
Web: www.beti.com.tr

2009