CAN (CONTROL AREA NETWORK) ÜZERİNDEN PIC PROGRAMLAMA

İlker ÜNAL

YÜKSEK LİSANS TEZİ
ELEKTRONİK BİLGİSAYAR EĞİTİMİ A.B.D
ISPARTA - 2006

T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

CAN (CONTROL AREA NETWORK) ÜZERİNDEN PIC PROGRAMLAMA

İlker ÜNAL

YÜKSEK LİSANS TEZİ
ELEKTRONİK BİLGİSAYAR EĞİTİMİ
ANABİLİM DALI
ISPARTA - 2006

Fen Bilimleri Enstitüsü Müdürlüğüne;

Bu çalışma jürimiz tarafından ELEKTRONİK BİLGİSAYAR EĞİTİMİ ANABİLİM DALI' nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Başkan :.....

Üye :.....

Üye :.....

ONAY

Bu tez ..../..../2006 tarihinde Enstitü Yönetim Kurulunca belirlenen yukarıdaki jüri üyeleri tarafından kabul edilmiştir.

..../..../2006
Prof. Dr. Çiğdem SAVAŞKAN
S.D.Ü. Fen Bilimleri Enstitüsü Müdürü

İÇİNDEKİLER

Sayfa No

CAN (CONTROL AREA NETWORK) ÜZERİNDEN PIC PROGRAMLAMA

İlker ÜNAL

Endüstride kullanılan elektronik modüller, birbirleri ile ve merkezi sunucu bilgisayar ile, özellikle, kontrol ve veri mesajlarının transfer edildiği ,endüstriyel ağ protokolü kullanarak haberleşmektedirler. CAN (Control Area Network) bu protokoller arasında en yaygın olarak kullanılanıdır. Bu çalışma, endüstriyel ortamlarda elektronik sistemlerin kontrolü için kullanılan mikro denetleyicilerin kontrolünü ve programlanmasını sunucu bilgisayar aracılığı ile yapılması amacını taşımaktadır.

Endüstriyel ağ sistemlerinde, tek bir ağ kablosu üzerinde birçok denetleyici modül bulunmaktadır. Bu modüllerin ihtiyaç duyulduğunda programlanması gerekmektedir. Programlanacak olan modüllerin sistem üzerinden çıkarılmadan programlanması üretimin aksamaması açısından çok önemlidir. CAN protokolü yardımı ile kontrolcü modüllerin programlama ve çalışmalarını izleme işlemleri mümkündür.

Bu sistemin gerçekleşmesi için, iki önemli noktanın oluşturulması gerekmektedir. Birincisi donanım olarak prototip setinin tasarlanması, ikincisi ise prototip setinin yönetilmesi için sunucu programının hazırlanmasıdır. Öncelikle PIC 18F458 mikro denetleyiciyi programlayacak olan sistemin donanımsal devresi gerçekleştirilmiştir. Daha sonra, bu donanımsal yapı üzerinden mikro denetleyicinin programlanabilmesi için Visual Basic dili kullanılarak görsel bir program yazılmıştır.

Çalışmanın ikinci bölümünde, CAN Protokolünün yapısı, kullanılan PIC mikrodenetleyicisi, PCI CAN Kartının yapısı ve Heksadesimal Dosyalar incelenmiştir. Üçüncü bölümde, uygulamaya esas teşkil edecek olan donanımsal ve yazılımsal yapılar anlatılmıştır. Son bölümde ise uygulama sonuçları yorumlanmıştır. Çalışmanın EK bölümünde ise, kullanılan dosyalar ve programlar sunulmuştur.

ANAHTAR KELİMELER: CAN Protokolü, PIC 18F458, Mikrodenetleyici Programlama

PIC PROGRAMMING VIA CAN (CONTROL AREA NETWORK)

İlker ÜNAL

Electronic modules which are used in industry communicating each other and central computer,especially, by using industrial network protocol on which control and data messages transmitted. CAN is the most common one among the these protocoles. This study aims to control and programming the microcontrollers used to control the electronic systems in industrial conditions by the server computer.

There are many controler modules ona single network cable in industrial network systems. These modules must be programmed in need. Programming the modules which will be programmed, without removing them from the system is very important for not stop process of production. Programming and controlling the modules is possible by using CAN protocol.

To realize this system, two important points must be created. The first one is designing the hardware prototype module and the second one is preparing server programme to manage the prototype module. Firstly, hardware circuit of system which will programme the PIC 18F458 microcontroller created. Then, Visual Basic programme prepared for programming PIC 18F458 microcontroller.

In the second chapter, structure of CAN protocol, PIC microcontroller used in system, structure of PCI CAN card and hexsadecimal files are presented. In the third chapter, hardware and software structures are explained. In the last chapter, the results of training are discussed. Files and programmes which are used in the system are given in the appendix.

KEY WORDS: CAN Protocol, PIC 18F458, Programming Microcontroller.

TEŞEKKÜR

Bu çalışmanın gerçekleştirilmesinde, bilgi, deneyim ve katkılarından dolayı danışman hocam Sayın Doç. Dr. Akif Kutlu' ya, çalışmanın uygulama safhasında fikirleriyle bana yardımcı olan arkadaşlarım Hüseyin Fidan ve Ünal Şanlı' ya teşekkürü borç bilirim.

İlker ÜNAL

SİMGELER DİZİNİ

ABS	Antilock Braking System
ACK	Acknowledge
ASCII	American Standard Code for Information Interchange
CAN	Control Area Network
CRC	Cyclic Redundancy Check
CSMA/CD+CR	Carrier Sense Multiple Access With Collision Dedection and Collision Resolution
ISO	International Standarts Organization
MAB	Message Assembly Buffer
OSI	Open System Interconnect
PCI	Peripheral Component Interconnect
PIC	Peripheral Interface Controller
REC	Recive Error Counter
RTR	Remote Transmit Request
SAE	Society of Automative Engineers
TEC	Transmit Error Counter
UTP	Unshileded Twisted Pair

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

CAN (Control Area Network) Protokolü, 1980' lerin ortalarında Alman otomotiv sistemleri üreticisi olan Robert Bosch tarafından, otomotiv uygulamalarında güçlü bir seri data iletiminin oluşturulması amacıyla ortaya çıkarılmıştır. Protokolün çıkış amacı, otomobillerin daha emniyetli, güvenli hale getirilmesi ve yakıt tasarrufunun sağlanabilmesiydi. CAN Protokolü, 1993 yılında ISO tarafından uluslararası bir standart olarak kabul edilmiştir. 1995 yılında SAE (Society of Automative Engineers) dizel motor uygulamalarında CAN Protokolünü standart olarak kabul etmiştir. CAN Protokolü ortaya çıkışından bu zamana kadar endüstriyel otomasyon sistemlerinde ve otomobil / kamyon uygulamalarında çok geniş bir popülerite kazanmıştır. Diğer birçok network çözümleri için güçlü avantajlar getirmiş ve tıbbi ekipmanlar, test cihazları ve mobil makinelerin yapımı gibi alanlarda büyük kolaylıklar sağlamıştır.

CAN Protokolü, asıl olarak otomotiv sistemleri için araçlar içerisindeki elektronik elemanların, seri bir bus üzerinden tek bir merkezi yönetici elemana data göndermesi prensibine göre çalışan bir sistemdir. Mercedes, Bmw, General Motor gibi büyük araba üretici firmaları araçlar içerisindeki elektronik sistemlerin haberleşmesinde CAN protokolünü kullanmaktadırlar. Örneğin, ABS (Antilock Braking System) fren sistemi, araçların yolda kaymasını engelleyen sistemler gibi kritik zamanlı sistemlerin tek bir merkezden yönetilebilmesini sağlar. Ayrıca, lambalar, kapılar, camlar, sıcaklık bilgileri gibi kritik olmayan bilgilerin de aynı merkezden kontrolüne olanak tanır. Birçok sistemin tek bir merkezi yönetici tarafından idare edilmesi doğal olarak sistem içerisinde birçok kablonun kullanılmasına sebep olmaktadır. CAN protokolü, kullandığı bus yapısı sayesinde kalabalık kablolama yapısını ortadan kaldırır. Çift sarmal bir kablo yardımıyla bile birçok elemanın birbirleri ile haberleşmesine olanak sağlar.

CAN Protokolü, son zamanlarda otomotiv sektöründe olduğu gibi endüstri sektöründe de yaygın olarak kullanılmaktadır. Örneğin, tarımsal makineler, tıbbi makineler ve otomasyon sistemleri içerisine CAN haberleşme protokolü girmiştir.

Tekstil üretim sistemleri, paketleme kontrol sistemleri, robot kontrol sistemleri gibi daha birçok sistem içerisinde CAN protokolü popülaritesini arttırmıştır. Bir çok büyük entegre üretici firmalarının CAN entegrelerini üretmesi ve fiyatlarının da ucuz olması CAN teknolojisinin geleceğini de garanti altına almaktadır.

CAN Protokolü, yüksek hızda bir seri ara yüze sahip olması, ucuz iletim hatını kullanması, kısa data uzunluğuna sahip olması, hızlı etkileşim zamanlarına sahip olması, çoklu master ve peer to peer haberleşme olanağı tanıması gibi üstün özellikleri sebebiyle kullanıcılara büyük avantajlar getirmektedir.

İçerisinde CAN modülü bulunan ilk silikon entegre 1989 yılında Intel Corp. tarafından piyasaya çıkarılmıştır. Bu tarihten itibaren, Siemens, Motorola, Philips ve Microchip gibi büyük firmalar CAN entegrelerini üretmeye başlamışlardır. Bu firmalardan biri olan Microchip firması, 2005 yılında toplam 200 milyona yakın CAN modüllü entegre satarak üretici firmalar arasında ikinci sıraya oturmuştur. Bu sebepten dolayı, tezimizin uygulama kısmında kullanılacak olan mikrodenetleyici, Microchip tarafından üretilen PIC 18F458 olarak seçilmiştir.

8 bitlik yapıda olan PIC 18F458 mikro denetleyici, 32768 KBayt program belleği, 1536 Bayt RAM bellek ve 256 Bayt Data EEPROM' a sahiptir. Bu mikredenetleyicinin CAN protokolü kullanılarak, bir CAN hattı üzerinden programlanması için bu zamana kadar herhangi bir çalışma yapılmamıştır. Yapılan tek çalışma entegre içerisine dosya yüklemek için kullanılan bootloader programının seri port üzerinden yüklenmesi ile ilgilidir.

İçerisinde CAN modülü bulunan bir mikredenetleyicinin CAN protokol yapısı kullanılarak programlanması için iki temel noktanın oluşturulması gerekir. Bunlar, bilgisayar ile mikrodenetleyiciyi birbirlerine bağlayacak olan elektronik devre yapısı ve bu yapı içerisinde çalışacak olan programlardır. Elektronik devre yapısı, bilgisayar, bilgisayarın PCI (Peripheral Component Interconnect) slotuna takılı olan bir CAN Kartı, CAN Kablosu, MCP2551 Transceiver (Alıcı – Verici) ve PIC 18F458 mikrodenetleyiciden oluşmaktadır. Elektronik devre, bilgisayara takılı olan

CAN Kartının CAN portundan çıkan CAN mesajlarını UTP (Unshileded Twisted Pair) tipi bir CAN kablosu üzerinden mikrodenetleyicinin CAN girişlerine aktarmak için tasarlanmıştır. CAN Kablosu üzerinden gelen sinyallerin, mikrodenetleyicinin anlayabileceği lojik seviyeye çevrilmesi amacıyla, CAN Kablosu ve mikrodenetleyici arasında bir Transceiver kullanılmıştır. Tasarlanan elektronik deverenin, mikrodenetleyicinin programlanmasında kullanılabilmesi için de iki tane program yazılmıştır. Bu programlardan bir tanesi bilgisayar üzerinde diğeri ise mikrodenetleyici üzerinde çalışmaktadır.

Bilgisayar üzerinde çalışan ve CAN Modül programlayıcısı ismi verilen programın amacı, programlanacak olan dataları hazırlayıp bilgisayarın CAN kartı üzerinden CAN mesajları halinde mikrodenetleyiciye göndermektir. CAN Modül programlayıcısı, programlanacak olan heksadesimal dosya içerisindeki data alanlarını bayt bayt alarak bir text dosya içerisine aktarmaktadır. Daha sonra, oluşan text dosya içerisindeki her 1 baytlık data CAN mesajı haline getirilerek mikrodenetleyiciye gönderilmektedir.

Mikrodenetleyici üzerinde çalışacak olan program ise, CAN Modül programlayıcı programı tarafından gönderilen mesajları alarak mikrodenetleyicinin ilgili bellek alanlarının programlanabilmesi amacıyla tasarlanmıştır. Bellek programlayıcı olarak adlandırılan bu program, öncelikli olarak paralel bir programlayıcı tarafından mikrodenetleyici içerisine yüklenmiştir. Program, gelen mesajların data alanı içerisinde bulunan 1 baytlık dataları alarak bir dizi içerisine aktarmaktadır. Her alınan 64 mesajdan sonra dizi içerisinde toplanmış olan datalar ilgili bellek alanı içerisine kaydedilmektedir. 64 bayttan sonra kayıt yapılmasının sebebi, kullanılan mikrodenetleyicinin yapısıyla alakalıdır. Çünkü çoğu mikrodenetleyici içerisine yapılacak olan yazma işlemi 64 baytlık bir işlemdir.

Bu çalışmada, CAN protokolünün mesaj yapısı kullanılarak, heksadesimal dosyalardan alınan bilgilerin, tasarlanan bir program vasıtasıyla CAN mesajlarına çevrilmesi ve CAN Bus üzerinden, mikrodenetleyiciye gönderilerek, yine tasarlanan bir program yardımıyla mikrodenetleyicinin programlanması amaçlanmıştır.

2. KAYNAK BİLGİSİ

2.1. Can Protokolünün Temelleri

CAN (Control Area Network) protokolü, güçlü ve hızlı bir seri haberleşme imkanı sağlaması sebebiyle birçok alanda kullanılmaktadır. Yaygın olarak otomotiv ve otomasyon alanlarında kullanılan CAN protokolü, hem kullanım kolaylığı hem de ucuz maliyeti ile günümüzde popülaritesini artırmıştır (Bosch, 2006).

CAN protokolü, OSI (Open Systems Interconnection) haberleşme modelini kısmen kullanmaktadır. Bu model Şekil 2.1.' de gösterilmiştir. Fakat bu modelin en alttaki iki katmanı sabit olarak aynen CAN tarafından kullanılmakta olup, diğer katmanlar üst seviyeli protokoller yardımı ile kullanılmaktadır (Richards, 2002).

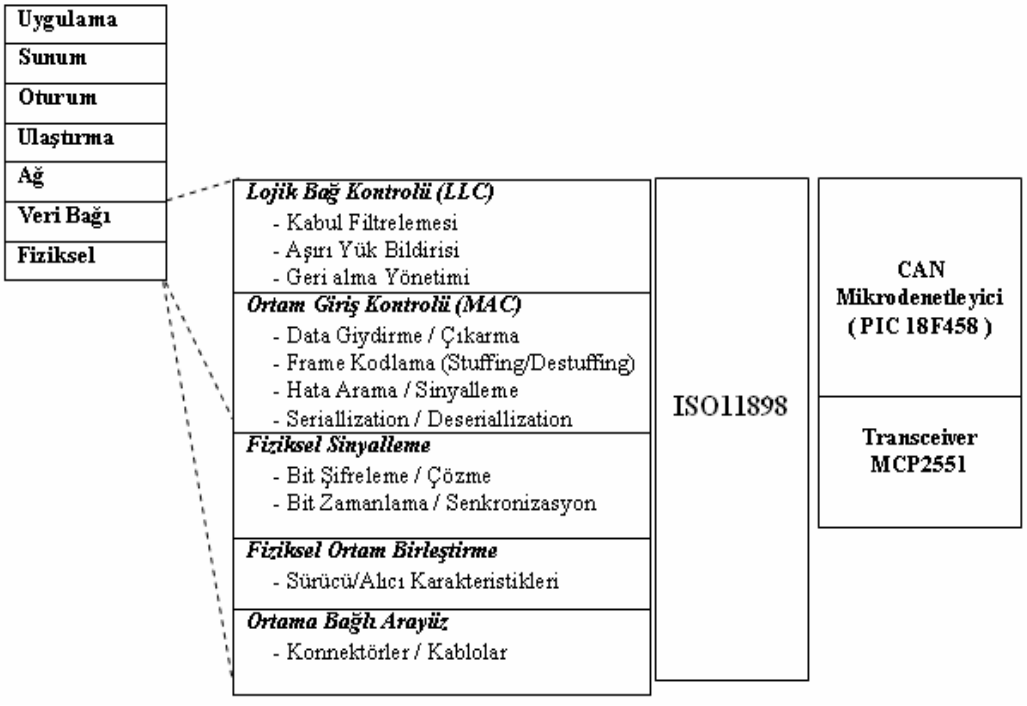
Şekil 2.1. OSI Haberleşme Modeli (OSI Model, 2004)

Haberleşme topolojisi olarak Hat, Yıldız, Halka topolojilerinden herhangi birini kullanmasına rağmen, Hat topolojisi yaygın olarak kullanılmaktadır. Mesajlaşma işlemlerinde CSMA / CD+CR (Carrier Sense Multiple Access With Collision Detection And Collision Resolition) yapısını kullanan CAN protokolü, mesaj çatışmalarında 'Arbitration' yöntemini kullanmaktadır. CAN mesajları, sistem içerisinde bağlı olan tüm modüllere gönderilmesi sebebiyle mesaj tabanlı bir protokoldür. Fiziksel iletim ortamı olarak çift sarmal kablo, koaksiyel kablo veya

fiber optik kablolardan herhangi birini kullanabilmektedir. Fakat hız ve güvenliğin istendiği ortamlarda, fiber optik kablo kullanımı tercih edilmektedir. CAN protokolü, maksimum 1 Mbit/saniye haberleşme hızına sahiptir. Tabii ki bu hız farklı iletim mesafelerine göre değişebilmektedir.(Tindell vd., 1995)

2.1.1. Can Haberleşme Katmanları

Çoğu network uygulamaları, sistem oluşumları için kalıpsal yaklaşımlar izlemektedirler. Bu şematik yaklaşım, farklı üreticilerin farklı ürünleri arasında birbirleriyle çalışabilme özelliğini sağlamaktadır. Bahsedilen bu kalıpsal yaklaşım ISO (International Standarts Organization) tarafından bir standart haline getirilmiştir. OSI (Open Systems Interconnection) adı verilen bu standart, network uygulamalarında referans model olarak kullanılmaktadır (Richards, 2002). CAN protokolünün kendisi bu referans modelin en alttaki iki katmanını kullanmaktadır. CAN protokolünün referans modeli Şekil 2.2.' de gösterilmiştir.



Şekil 2.2. CAN Protokolünün Referans Modeli (Richards,2002)

CAN haberleşmesi 3 katmandan oluşmaktadır. İlk iki katman OSI referans modelinin aynısıdır. Üçüncü katman ise uygulama katmanıdır. Ancak ilk iki katman kendi içerisinde 3 alt katmana ayrılmıştır.

- Fiziksel Katman
- Transfer Katman
- Nesne Katman

Fiziksel katman, iletim ortamının elektriksel yapısını, kablolama, sinyal zamanlaması ve CAN modülleri arasında CAN mesajlarının ortamdan iletilmesi işlemlerini yerine getirmektedir.

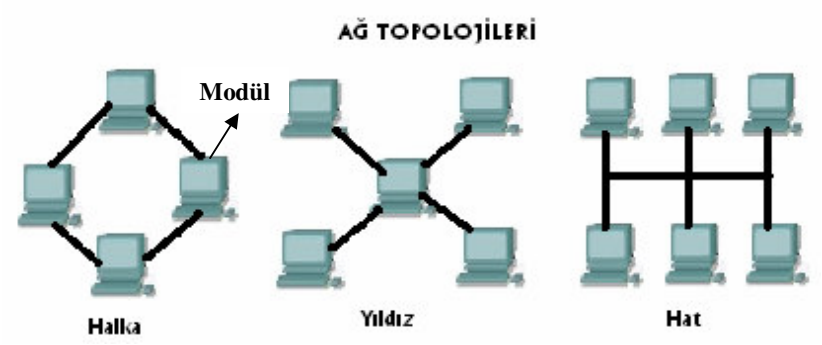
Transfer katmanı, hata tespiti, arıza hapsi, Arbitration, mesaj çerçeveleme, transfer hızı ve zamanlaması ve mesaj onaylama ve kabulü gibi işlemleri yerine getirmektedir.

Nesne katmanı, transfer ve uygulama katmanları arasında haberleşme işleminin sağlanması için tasarlanmıştır. Gerekli olan tüm mesajlar için filtreleme işlemini yapmasıyla beraber bağlı olan tüm donanımların uygulama katmanları için bir ara yüz oluşturur.

Son katman olan uygulama katmanı ise uygun haberleşme nesnesi, tesis, konfigürasyon ve CAN tabanlı ağların denetimini kullanarak, CAN tabanlı elemanların modellenmesi için servis sağlamaktadır. Aynı zamanda katman parametrelerinin tanımlanmasını ve ayarlanmasını sağlamaktadır.

2.1.2. Can Topolojisi

CAN topolojisi, network üzerindeki CAN modüllerini birbirine bağlayan kablonun yapısıdır. CAN protokolünde kullanılan topolojiler Hat, Halka veya Yıldız topolojilerinden biri olabilir (Şekil 2.3.).



Şekil 2.3. CAN Protokolünde Kullanılan Topolojiler

CAN Protokolünde kullanılan en yaygın topoloji tipi, Hat topolojidir (Richardson vd., 2001). Bu topolojide, her modül ağ içerisindeki mesajlaşmayı sağlayabilmek için tek bir seri kablo ile birbirlerine bağlıdır. İstasyonlar arasında CSMA/CD+CR erişim kuralları kullanılır. Bu kurallar, iki modül aynı anda mesaj gönderdiğinde ortaya çıkan çarpışmaları engellemek için kullanılır (Pazul, 1999).

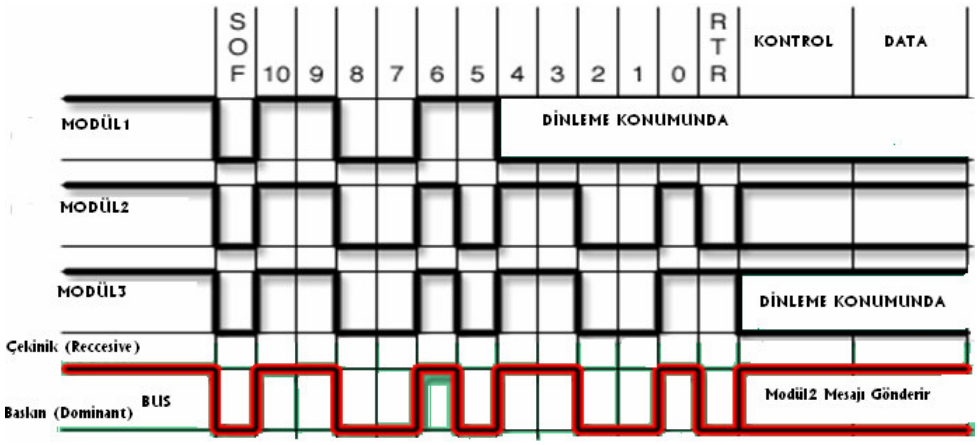
2.1.3. Carrier Sense Multiple Access With Collision Detection And Collision Resolition (CSMA/CD+CR) Ve Arbitration İşlemi

CAN protokolü, CSMA/CD+CR yapısında bir protokoldür. CSMA, ağda bulunan tüm modüllerin, bus üzerinden bir mesajı göndermeye çalışmadan önce bir periyotluk boş işlem zamanının izlenmesi anlamına gelmektedir. Boş işlem periyodu oluştuğu anda, bus üzerinde bulunan tüm modüller, eşit haklara sahip hale gelirler. CD ise, ağda iki modülün aynı zamanda mesaj göndermeye başlaması neticesinde oluşan çatışmanın algılanmasıdır. CR ise, bu çatışmanın çözümlenmesi anlamına gelmektedir. Çatışmanın çözümlenmesinde kullanılan yöntem ise 'Arbitration' yöntemidir (Lawrenz, 1997; Hopkins, 2003).

CAN protokolü, bit seviyesinde bozucu olmayan Arbitration metodunu kullanmaktadır. Bunun anlamı, bir çatışma olayı meydana geldiğinde Arbitration tamamlandıktan sonra mesajın bozulmadan, orijinal haliyle kalabilmesidir.

Arbitration işlemi neticesinde,yüksek önceliğe sahip olan mesaj gecikmeksizin ve bozulmaksızın gönderilebilmektedir.

Bit seviyesinde arbitration işleminin gerçekleşebilmesi için iki önemli noktanın oluşması gerekmektedir. Birincisi, sistemdeki lojik durumların baskın (dominant) ve çekinik (recessive) olarak tanımlanması gerekir (Tindell vd., 1994). İkincisi ise, bus üzerinde aktuel olarak görünen gönderme girişimlerinin lojik durumlarının nasıl olduğunun mesaj gönderen modül tarafından gözetlenebilmesidir. CAN protokolü, lojik 0 seviyesini baskın, lojik 1 seviyesini ise çekinik olarak tanımlamaktadır. Baskın bit durumu, her zaman çekinik bit durumunun üzerinde arbitration işlemini kazanacaktır. CAN mesaj çerçevesi (Frame) içerisinde bulunan Message Identifier' deki (arbitration işleminde kullanılan alan) en düşük değerden dolayı, gönderilecek olan mesaj en yüksek öncelikteki mesaj olacaktır. Örnek olarak, aynı zamanda iki modül mesaj göndermeye çalıştıklarında, öncelikle bus üzerindeki aktüel lojik durumu anlayabilmeleri için bus' ı izleyeceklerdir. Düşük öncelikli mesaj, bir noktadan sonra çekinik bir bit göndermeyi deneyecektir. Bu noktada, bu biti gönderen modül arbitration işlemini kaybedecek ve aniden mesaj gönderme işlemini durduracaktır. Yüksek önceliğe sahip mesaj, gönderme işlemi bitene kadar bus üzerinde istenen yerlere gidecektir (Kutlu, 2004). Arbitration işlemini kaybeden modül bir sonraki boş işlem periyodunu bekleyecek ve daha sonra mesajını tekrar gönderecektir. Bu durum Şekil 2.4' de gösterilmiştir.



Şekil 2.4. Arbitration işlemi

2.1.4. Mesaj Tabanlı Haberleşme

CAN protokolü, mesaj tabanlı bir protokoldür. Mesaj gönderme işlemi modüllerin adreslerine göre yapılmamaktadır(Farsi, vd.,1999). Mesajın içerisinde içerik ve öncelik bilgisi vardır. Gideceği modülün adresi yoktur. Sistem içerisinde bulunan her modül, bus üzerinden gönderilen her mesajı alır. Bu noktadan sonra, modüller gelen mesaja bakarak kendilerine ait ise kabul ederler, değil ise reddederler (Ekiz, vd., 1996). Bir mesaj, sistemin dizayn şekline göre ya belirli bir modüle ya da birçok modüle gönderilebilir.(Lawrenz, 1997; Hopkins, 2003)

Mesaj tabanlı haberleşmede, sistem içerisine yeni bir modül bağlandığında diğer modüllerin takılan bu modülü tanımaları için yeniden programlanmalarına gerek yoktur. Yeni modül, sisteme takılır takılmaz diğer modüllerden gelecek olan mesajları almaya başlar ve kendi üzerindeki programa ve gelen mesajın tanımlama bilgisine göre mesajı kabul eder veya reddeder (Ekiz, vd., 1996).

2.1.5. İletim Ortamı

İletim ortamı mesajların gönderileceği fiziksel yoldur. Yaygın olarak kullanılan iletim ortamları, paralel ve çift sarmal kablo, koaksiyel veya fiber optik kablolardır. İletim ortamının seçimi, yapılacak olan uygulamanın güvenlik şartlarına ve parasal imkanlara bağlıdır. Fiber optik kablo kullanımı ideal bir seçimdir ve bazen de gereklidir. Ağda çalışan modüllerin elektriksel olarak birbirlerinden izole edilmesi ve kısa devre veya açık devre gibi istenmeyen arızaların oluşmasının engellenmesi fiber optik kablo ile sağlanabilir. Fakat, fiber optik kablo diğer kablolara göre pahalı bir seçimdir. Hangi kablo tipi seçilirse seçilsin, yapılacak olan uygulama ortamı için iletim yolu tamamen tek bir tip kablo olmalıdır (Wei vd., 2005). CAN protokolü maksimum 1 Mbit/sec haberleşme hızına sahiptir. Tabii ki bu hız farklı iletim mesafelerine göre değişebilmektedir (Çizelge 2.1.).

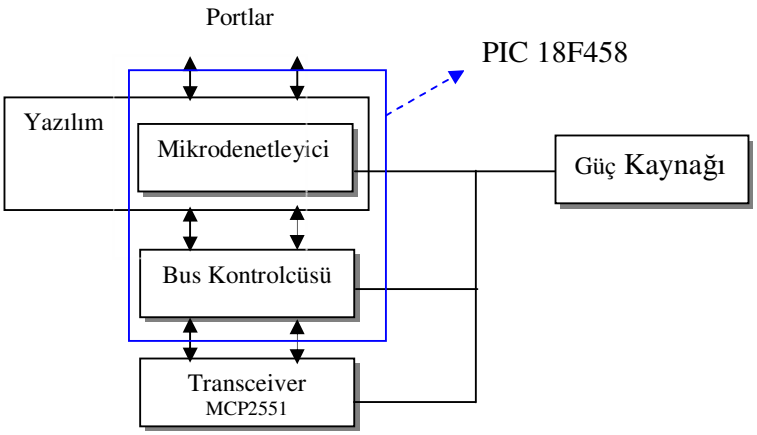
Çizelge 2.1. İletim Mesafelerine Göre Bus Hızları

Bus Uzunluğu (metre)	Maksimum Bit Oranı (bit/s)
40	1 Mbit/s
100	500 kbit/s
200	250 kbit/s
500	125 kbit/s
6000	10 kbit/s

2.1.6. Can Modülü

CAN Modülü, CAN hattı üzerinden gelen mesajlara göre hareket eden elektronik yapılardır. Donanımsal olarak bir CAN ağının temel elemanlarıdır. Kullanıcı tabanlı uygulama yazılımları bu modüllerin üzerinde çalışır. Basit bir CAN modülü ile, iletişimin tüm kurallarını yerine getirebilmek için 3 temel yapının oluşturulması gereklidir. Bunlar; elektronik modüller, modül kontrolü için yazılımlar ve fiziksel iletişim ortamıdır (Şekil 2.5). Sistemde kullanılan elektronik bir modül içerisinde aşağıdaki yapılar bulunur.

- Fiziksel Bus Ara yüzü (Transceiver)
- Bus Kontrolcüsü (Bus Controller)
- Mikro denetleyici (Microcontroller)
- Uygulama Arayüzü (Application Interface)
- Güç Kaynağı (Power Supply)



Şekil 2.5. CAN Modülü

Fiziksel ara yüz, bus üzerindeki elektriksel sinyali lojik seviyeye çevirmek için kullanılır. Bu eleman yardımıyla, CAN kontrolcüsü ve fiziksel bus arasında bir bağlantı sağlanır. CAN entegresi olarak bilinen bus kontrolcüsü, kullanıcı bilgilerini CAN mesaj çerçevesi içerisinde iletim kanalı üzerinden gönderilen fiziksel bitlere çevirir (Microchip Inc., 2003). Piyasada 3 çeşit CAN entegresi bulunmaktadır. Bunlar;

- Stand – Alone Entegreler
- Serial Linked Entegreler
- Bütünleşik Entegreler

Stand – Alone entegreler, data ve adres hatları üzerinden sistemi yöneten mikro denetleyiciye bağlanırlar. Bütünleşik yapıda olan entegreler içerisinde hem sistemi yöneten mikrodenetleyici hem de CAN kontrolünü yapan CAN modülü bulunmaktadır. Serial – Linked tipi entegreler yaygın olarak kullanılmazlar (Farsi vd., 1999). Bu tip entegrelerin programlanabilmesi için sistem içerisinde bir başka mikro denetleyiciye ihtiyaç vardır.

Mikrodenetleyici üzerindeki yazılım, kullanıcı tanımlı uygulamaları çalıştırmak için kullanılmaktadır. Portlar ve CAN entegresi arasındaki data yakalama işlemini kontrol eder. Data yakalama işlemi süresince, mesajların tamponlanma metoduna göre CAN kontrolcüleri iki sınıfa ayrılmaktadır. Eğer mesajlar CAN kontrolcüsünün Ram' inde tutulacaksa buna FullCan adı verilir. Eğer mesaj mikrodenetleyicinin belleğinde tutulacak ise buna BasicCan adı verilir (Farsi vd., 1999). BasicCan, mesaj yakalama işleminde FullCan' e göre daha esnek ve hızlıdır (Pazul, 1999).

2.1.7. CAN Mesaj Çerçeveleri

CAN protokolü içerisinde 4 farklı mesaj tipi bulunur (Hopkins, 2003). Bunlar;

- Data Frame
- Remote Frame
- Error Frame
- Overload Frame

Bu mesaj çerçevelerinden, data frame içerisinde data, diğerlerinde ise kontrol amaçlı mesajlar bulunur.

2.1.8. Data Frame

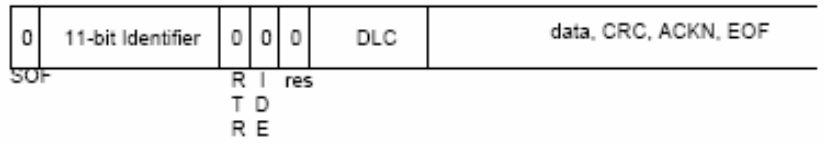
CAN tarafından tanımlanmış olan mesaj hakkında ekli bilgi sağlayan alanları içerir. İki tip data frame bulunur. Bunlar;

- Standart Data Frame
- Extended Data Frame

Her iki frame içerisinde ortak olarak farklı amaçlar için kullanılan 7 adet alan bulunmaktadır. Bunlar;

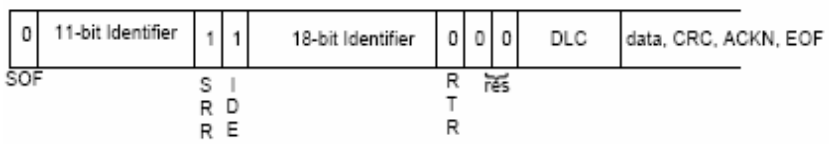
- Başlangıç Biti (Start of Frame)
- Arbitration Alanı (Arbitration Field)
- Kontrol Alanı (Control Field)
- Data Alanı (Data Field)
- CRC Alanı (Cyclic Redundancy Check Field)
- ACK Alanı (Acknowledge Field)
- Frame Sonu (End of Frame)

Standart frame içerisinde 12 bitlik arbitration alanı bulunur. Bu 12 bitten 11' i mesajı tanımlamak için, 1 bit ise RTR bitidir. Bu bit data frame' in uzaktaki bir modülden istekte bulunabilmesi için kullanılır. Şekil 2.6.' da standart bir data frame gösterilmektedir (Zuberi ve Shin, 1995).



Şekil 2.6. Standart Data Frame

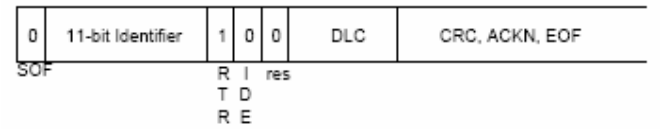
Extended Data Frame içerisinde 32 bitlik arbitration alanı bulunur. 32 bitin 29' u mesajı tanımlamak için kullanılır.1 bit IDE bitidir. Bu bit data frame' in extended formatta olduğunu göstermek için kullanılır. 1 bit SRR biti bulunur. Bu bit kullanılmaz. 1 bit de RTR biti bulunur. Bu bit frame' in standart mı yoksa extended mi olduğunu göstermek için kullanılır (Zuberi ve Shin, 1995). Şekil 2.7.' de Extended Data Frame' i gösterilmektedir.



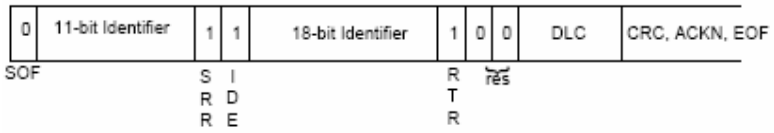
Şekil 2.7. Extended Data Frame

2.1.9. Remote Frame

Data frame içerisinde bulunan RTR biti, bir modülün başka bir modülden bilgi istemesi durumunda kullanılmaktadır. Remote frame içerisinde data alanı bulunmamaktadır. Bir remote framein yapısı, standart frame için Şekil 2.8' de ve extended frame için Şekil 2.9' da gösterilmiştir. Bir modül, remote frame gönderirken istekte bulunduğu data framein tanımlayıcı bilgisini koymalı ve RTR bitini lojik 1 yaparak göndermesi gerekmektedir (Lawrenz, 1997). Hem data hem de remote framelerinin arbitration alanında bulunan RTR biti, remote frame ile data frame arasındaki farkı oluşturmaktadır. Eğer RTR biti lojik 1 ise bu frame, remote frame, lojik 0 ise data frame olduğu anlaşılır.



Şekil 2.8. Standart Remote Frame

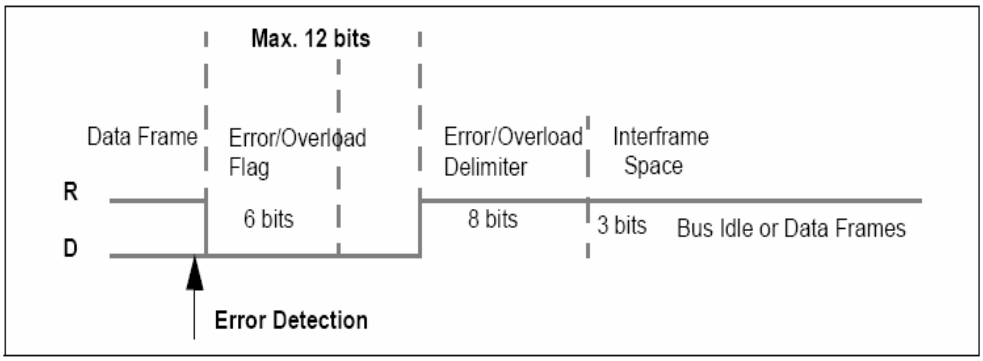


Şekil 2.9. Extended Remote Frame

2.1.10. Error Frame

Bu frame, bir mesaj gönderildikten sonra veya gönderim süresince bir hata oluştuğu zaman gönderilen kontrol amaçlı bir framedir. Şekil 2.10' da Error framein yapısı gösterilmiştir. Bir hata frame içerisinde 6 veya 12 adet aynı lojik seviyeli bit ve 8 adet hata sınırlama biti bulunmaktadır. Her aynı seviyeli 6 bitlik grup hata bayrağı olarak adlandırılmaktadır. Gönderici yada alıcı modül hata bayrağı gönderdiği zaman

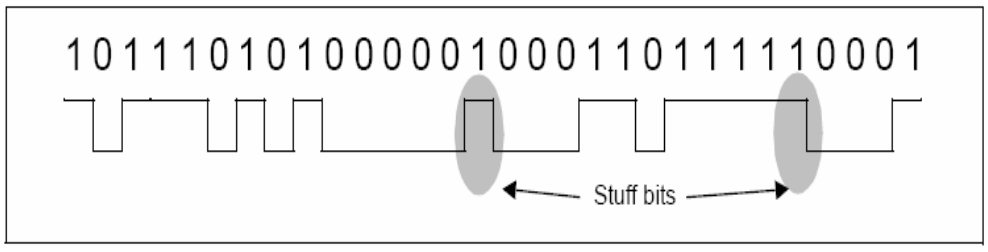
diğer modüller bit stuffing kuralının ihlal edildiğini anlar ve hepsi tekrar hata bayrağı gönderirler.



Şekil 2.10. Error / Overload Frame Yapısı

2.1.11. Bit Stuffing Metodu:

Bit stuffing metodu, CAN data yada remote frame içerisinde ardı ardına aynı lojik seviyeli 5 bit gönderildiği zaman 6. bitine ters lojik seviyeli bitin eklenmesidir (Nolte vd., 2001). Bu ters lojik seviyeli bitin eklenmesinin sebebi, örnekleme işlemi için çok fazla sayıda kenar kullanarak frame senkronizasyonunun sağlanmasıdır. Eklenen bu bit, alıcı tarafından ardı ardına aynı seviyeli 5 bitin gözlenmesinden sonra 6. bitin silinmesiyle kaldırılır. Şekil 2.11.' de bit stuffing işlemi gösterilmektedir.



Şekil 2.11. Bit Stuffing İşlemi

2.1.12. Overload Frame

Overload frame, alıcı durumdaki modülün bir sonraki gelecek olan mesaj için dahili çalışma zamanının yeterli olmaması ve Inter-Frame alanı içerisinde dominant (Lojik 0) bir bitin bulunması durumunda gönderilir. Şekil 2.10.' da Overload Frame yapısı gösterilmiştir.

2.1.13. CAN Hata Yapıları ve Modları:

CAN protokolü kullanılarak gerçekleştirilen haberleşme ortamlarında genelde karşılaşılabilecek 5 çeşit hata yapısı vardır. Aynı zamanda bu hatalara karşılık tespit edilen hata durumlarının sayısı ve tipine bağlı olarak modüller üzerinde oluşan 3 farklı hata modu mevcuttur. Tespit edilen hata yapıları;

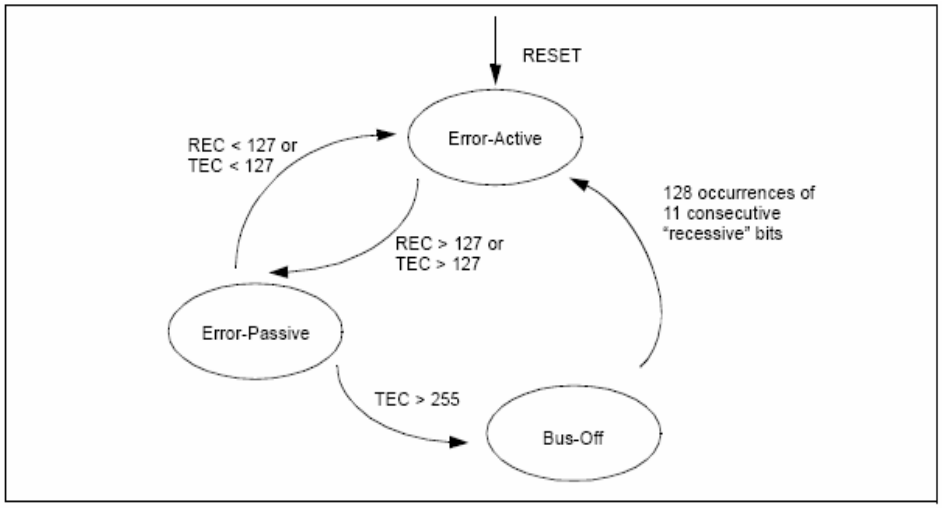
- CRC Error
- Acknowledge Error
- Form Error
- Bit Error
- Stuff Error

Data frame içerisinde bulunan CRC alanı içerisinde gönderilen mesajın toplam bit sayısı bulunmaktadır. Eğer herhangi bir modül gelen mesajın toplam bit sayısı ile CRC alanındaki sayı arasında farklılık görürse CRC hatası oluşur. Data frame içerisinde bulunan ACK Slot biti mesaj gönderilirken 1 yapılır. Eğer herhangi bir modül bu biti 0 olarak görürse Acknowledge hatası oluşur. Eğer herhangi bir modül CAN protokolünün içerisinde kullanılan mesaj framelerinin yapısında bir bozukluk tespit ederse Form hatası oluşur (Pazul, 1999). Eğer gönderici modül lojik 1 biti gönderirken bus üzerinde o biti lojik 0 olarak görürse yada tersi Bit hatası oluşur. Eğer mesaj frame içerisinde aynı seviyeli 6 bit tespit edilirse Stuff hatası oluşur.

Oluşan hata durumlarının sayısı ve tipine göre modüllerin üzerinde oluşan hata modları şunlardır;

- Error – Active
- Error – Passive
- Bus – Off

Hata modlarının tespiti için oluşan hataları sayan 2 tane sayıcı vardır. Bir tanesi gönderici modül için çalışan TEC (Transmit Error Counter) sayacı, diğeri ise alıcı modül üzerinde çalışan REC (Recive Error Counter) sayacıdır. TEC, mesaj gönderilirken oluşan hataları sayar, REC ise mesaj alınırken oluşan hataları sayar. Şekil 2.12.' de sayaçlar ve hata modları gösterilmektedir.



Şekil 2.12. TEC ve REC Sayaçları ve Hata Modları

Şekilden de anlaşılacağı üzere, Sistem resetlendiği andan itibaren aktif hata durumuna geçer. TEC ve REC sayıcılarının gösterdiği değer 127' nin altında ise modül aktif hatalı, üzerine çıkarsa pasif hatalı duruma geçer. TEC sayıcısı 255' in üzerine çıkarsa modül hattan çıkar (Bus Off) (Gaujal vd., 2005).

2.2. Microchip PIC 18F458 Mikrodenetleyici

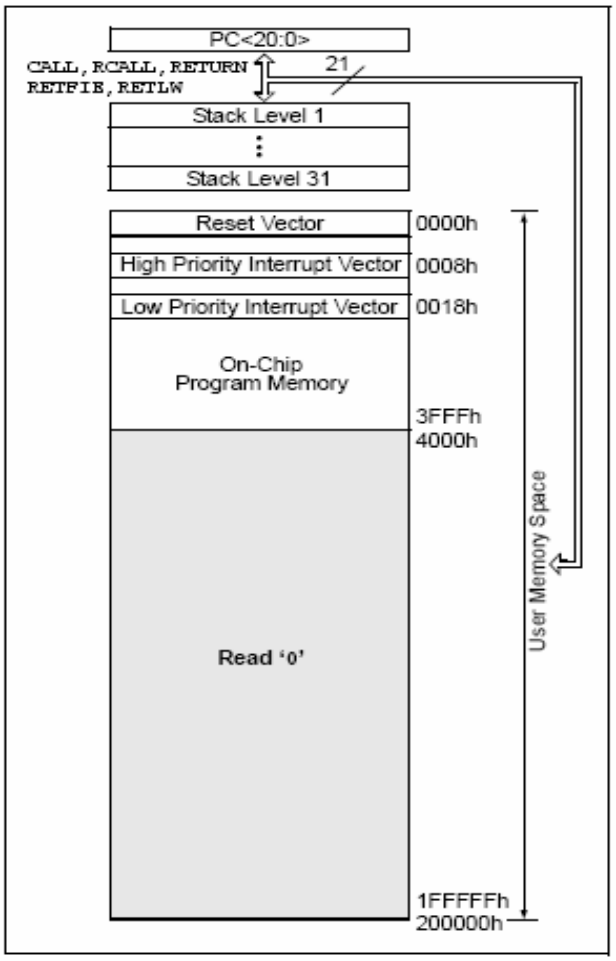
Microchip firması tarafından üretilen PIC 18F458 mikro denetleyici, içerisinde barındırdığı CAN modülü sayesinde bir çok CAN uygulamalarında kullanılmaktadır.

8 bitlik yapıda olan PIC 18F458 mikro denetleyici, 32768 KBayt program belleği, 1536 Bayt RAM bellek ve 256 Bit Data EEPROM' a sahiptir. Mikro denetleyiciye ait detaylı teknik bilgi Çizelge 2.2.' de verilmiştir (Microchip Inc., 2004).

Çizelge 2.2. PIC 18F458 Mikrodenetleyici Özellikleri

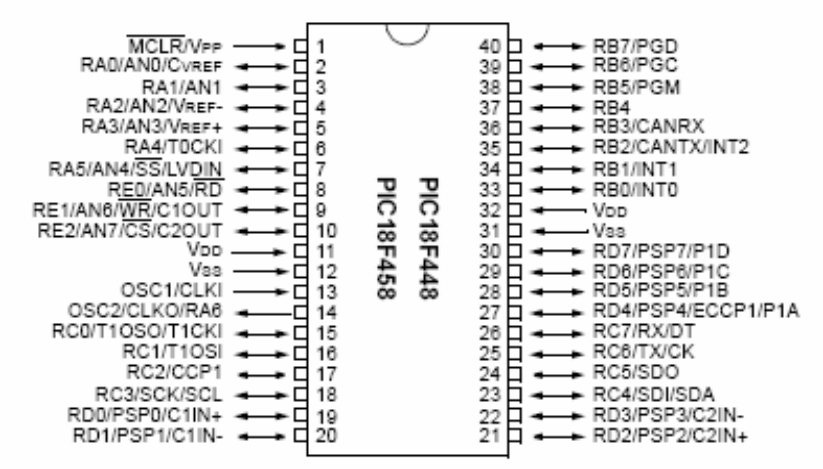
Özellikler		**PIC 18F458**
Çalışma Frekansı		Max.DC-40Mhz
Dahil Program Belleği	Bayt	32K
	Single-Word işlem	16384
Data Bellek		1536 Bayt
Data EEPROM Bellek		256 Bayt
Interrupt Kaynağı		21
I/O Portları		A,B,C,D,E Portları
Zamanlayıcı		4
Seri Haberleşme Kaynakları		MSSP,CAN, Adreslenebilir USART
Paralel Haberleşme		Var
10 Bit Analog-Dijital Çevirici		8 Kanallı Giriş
Analog Karşılaştırıcı		2
CAN Modülü		Var
İşlem Sayısı		75 İşlem
In-Circuit Seri Programlama (ICSP),LVP,HVP		Var

PIC 18F458 mikrodenetleyici, 21 bitlik program sayıcısı yardımı ile 2 Mbayt program belleği adresleyebilmektedir. Aynı zamanda 4 Kbayt data belleği adresleyebilme özelliğine sahiptir. Mikrodenetleyicinin program bellek yapısı Şekil 2.13' de gösterilmiştir.



Şekil 2.13. PIC 18F458 Program Bellek Yapısı (Microchip Inc., 2004)

Mikrodenetleyici, 8 bit data uzunluğuna sahiptir. Mikro denetleyici 40 bacaklı olup bacak bağlantıları Şekil 2.14.' de gösterilmiştir.



Şekil 2.14. PIC 18F458 Bacak Bağlantıları

2.2.1. PIC 18F458 CAN Modülü

PIC 18F458 mikro denetleyici içerisinde bulunan CAN modülü, seri bir arayüz olup, diğer çevre elemanları ve mikro denetleyicilerle haberleşme işlemlerinde kullanılmaktadır. Bu modül, gürültülü ortamlarda seri haberleşmenin sağlanabilmesi amacıyla tasarlanmıştır. CAN protokolü içerisindeki CAN 2.0A/B Pasif ve CAN 2.0B Aktif versiyonlarını destekleyen bu modül, Full CAN yapıya sahiptir (Microchip Inc., 2004).

Modül özellikleri;

- CAN 2.0A/B versiyonlarını destekler
- Standart ve Extended Data çerçevelerini destekler
- 0-8 Bayt data genişliğine sahiptir
- Programlanabilme hızı 1 Mbit/saniye
- Çift tamponlu alıcı mevcuttur
- 6 adet kabul filtresi bulunur ve bunlardan 2’ si yüksek öncelikli mesajlar için 4’ ü de düşük öncelikli mesajlar için kullanılmaktadır
- 2 adet kabul filtre maskesi vardır

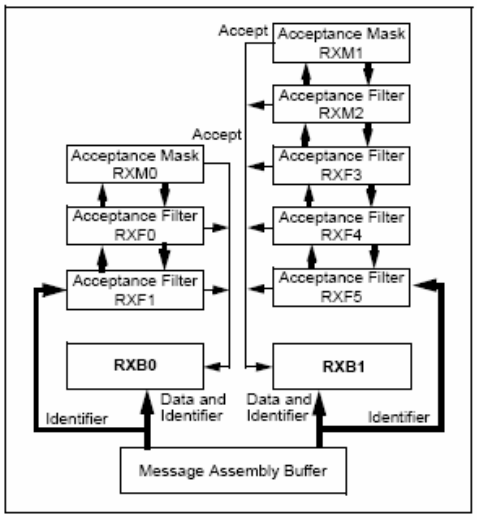
- 3 adet mesaj gönderme tamponu bulunmaktadır.

CAN modülü, protokol mekanizması olarak, mesaj tamponlama ve kontrol işlemlerini içerisinde barındırır. CAN protokol mekanizması, CAN bus üzerindeki gelen ve gönderilen mesajların tüm fonksiyonlarını idare ve kontrol etmektedir. Mesajlar, uygun data registerleri üzerine yüklenerek gönderilir. Mesajın durumu ve hataları bu registerlar okunarak kontrol edilmelidir. CAN bus üzerinde bulunan herhangi bir mesaj, hatalar için kontrol edildikten sonra uygun olan filtreden geçer ve 2 alıcı registerden biri üzerine yüklenir.

CAN modülü, PIC 18F458 mikro denetleyicinin RB3/CANRX ve RB2/CANTX/INT2 ayak bağlantıları yardımıyla CAN bus üzerine dahil edilir.

2.2.2. CAN Modülün Mesaj Kabulü:

PIC 18F458 mikrodenetleyicisi CAN mesajlarını almak için 2 tane alım tamponu kullanmaktadır. Ayrıca gelen mesajları alım tamponlarına ayrıştıran mesaj toplama tamponu olan MAB' da üçüncü bir alım tamponu gibi hareket etmektedir. Şekil 2.15.' de CAN modülünün mesaj alım tampon yapısı gösterilmektedir.



Şekil 2.15. Mesaj Alımı Tampon Yapısı (Microchip Inc.,2004)

Bu 3 tampondan biri olan MAB, bus üzerinden gelen ilk mesajları karşılar. Geriye kalan diğer iki tampon RXB0 ve RXB1, protokol mekanizmasındaki mesajın tamamını alırlar. Mikrodenetleyici, bir tampondan mesaj alırken diğer tampon ise gelecek olan yeni mesajı bekler veya üzerinde mesaj varsa diğer tamponun işi bittikten sonra mesajı mikrodenetleyiciye gönderir.

MAB gelen tüm mesajları kendi üzerinde bekletir. Bekleyen bu mesajlar, MAB tarafından, eğer bir kabul filtre kriteri ile karşılaşırsa diğer iki tampondan birine gönderilir.

Mesaj, tamponlardan herhangi birine gittiği zaman, uygun olan RXBnIF biti '1' olur. Bu bit, mikro işlemci tarafından, tampon üzerinden mesaj alındıktan sonra, tamponun bir sonraki gelen mesajları alabilmesi için '0' yapılır. Eğer bu aşamada RXBnIE biti '1' olarak tespit edilirse, geçerli bir mesajın mikrodenetleyici tarafından alındığını göstermek için bir kesme üretilir.

CAN modülünün mesaj kabulü ile ilgili işlemler mikrodenetleyinin içerisine yazılan kullanıcı programları vasıtasıyla gerçekleştirilir. Program içerisinde hangi mesaj alım tamponlarının kullanılacağı belirtilir. Ayrıca gelen mesajların kontrolü amacıyla kullanılan maske tamponlar kullanılır. Bu tamponlar gelen mesajın ID alanında bulunan değerleri üzerlerinde barındırırlar. Eğer gelen mesajın ID alanındaki değer ile maske tampon alanına yerleştirilmiş olan değer uyuşursa mesaj kabul edilir.

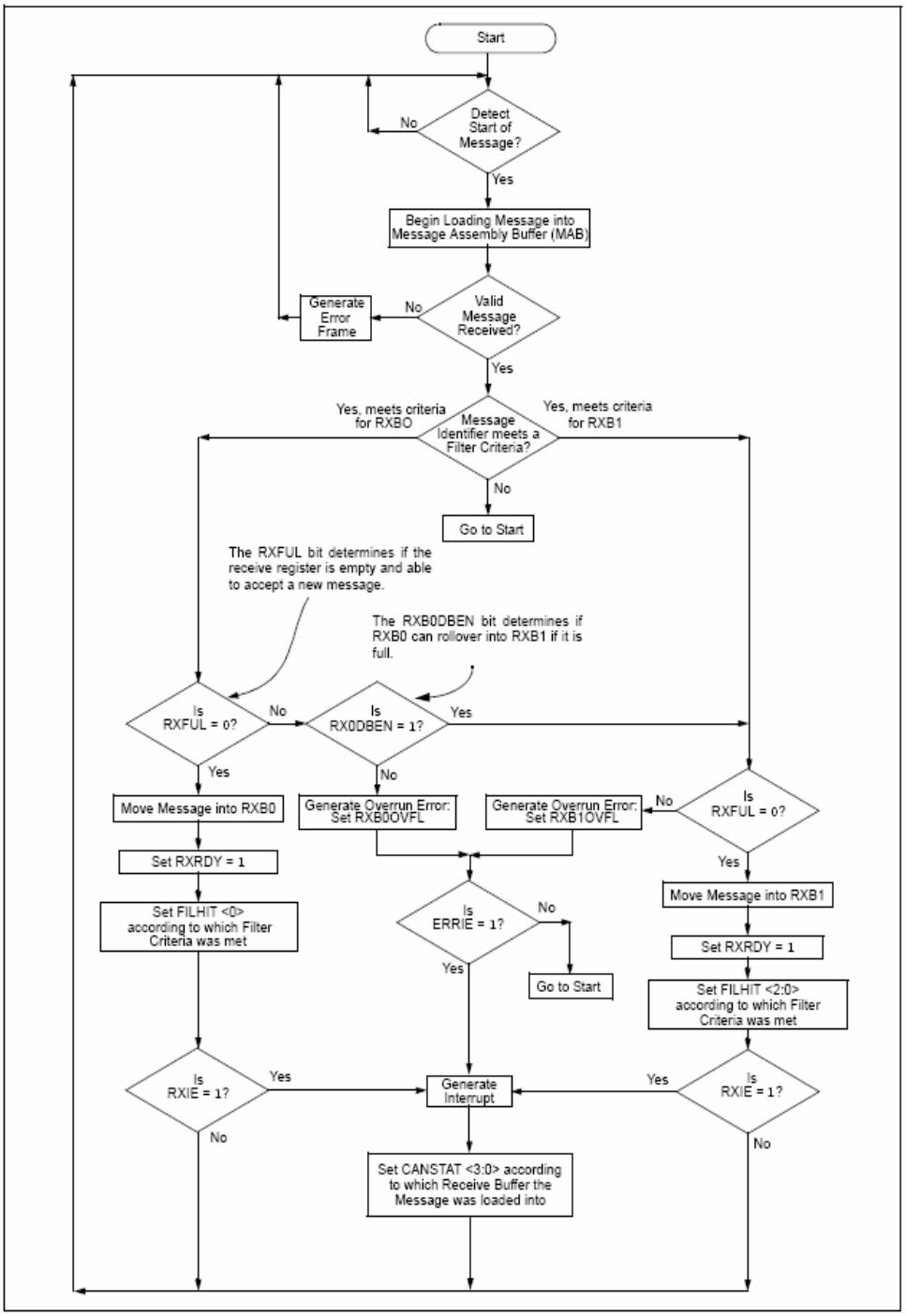
2.2.3. Modülün Mesaj Alım Önceliği:

RXB0 tamponu en yüksek öncelikli alım tamponudur ve 2 tane mesaj kabul filtresine sahiptir. RXB1 tamponu ise en düşük öncelikli alım tamponudur ve 4 tane mesaj kabul filtresine sahiptir. Kabul filtrelerinden düşük numaralı olanı, gelen mesajın en kısıtlayıcı ve en öncelikli olduğunu göstermek için karşılaştırma yapar. Bunun yanında, her bir tampon içerisinde programlanabilen birer mesaj kabul maskeleyicisi bulunur.

Bir mesaj alındığı zaman, RXBnCON kaydedicisinin 0 ve 3. bitleri arası mesajın hangi kabul filtresinde olduğunu ve uzak transfer isteği olup olmadığını gösterir. RXM bitleri özel alım modlarının ayarlanması için kullanılır. Normal olarak, bu bitler ‘00’ yapıldığında tüm geçerli mesajlar alınır. Eğer ‘01’ yada ‘10’ olursa, standart yada extended yapıdaki mesajlar kabul edilir. Eğer ‘11’ olursa tüm mesajlar, hata içerikli mesajlarda dahil olmak üzere kabul edilir.

CAN Protokolü mesaj tabanlı bir sistemdir. CAN Bus üzerindeki tüm modüller, bus üzerindeki tüm mesajları gözlerler. Bus üzerinden gönderilen mesajlar tüm modüller tarafından alınır. Modüllerin içerisinde yüklü olan yazılımlara göre, mesaj kabul edilir veya reddedilir. Modül, gelen mesajın tanımlayıcı alanı içerisinde bulunan değere göre mesajı kabul edip atmeyeceğine karar verir. Mikrodenetleyici içerisinde çalışan filtreler me maskeleyiciler, gelen mesajın, tanımlayıcı alanı içerisindeki değer ile karşılaştırılır. Eğer bir eşitlik sözkonusu ise mesaj kabul görür. Bu şekilde bir mesaj birden fazla modül tarafından da kabul edilebilir.

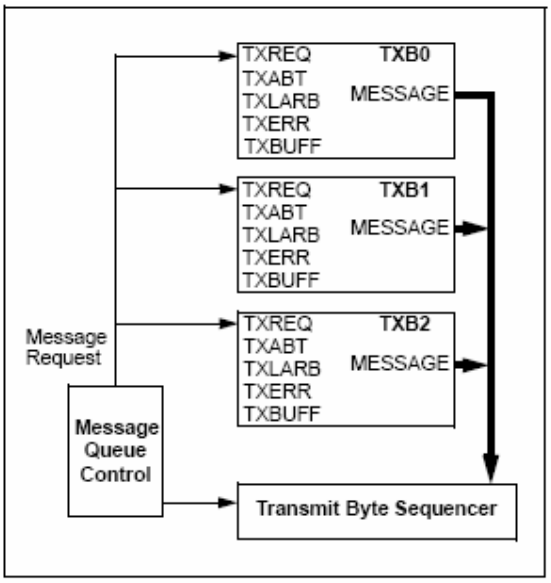
Sistemi tasarlayanlar, hangi filtre ve maskeleri kullanacağına yazılım içerisinde karar verirler. Çoğu mikrodenetleyici içerisinde birden fazla kabul filtresi bulunmaktadır. Bu filtrelerin yazılım içerisindeki ayarlamaları sayesinde gelen mesajların öncelikleri de belirlenebilir. Yani, mikrodenetleyici birden fazla farklı mesaj için öncelik belirleyebilir (Kavlico Corp., 2000). Şekil 2.16.’ de bir mesajın gönderimine ilişkin akış şeması gösterilmektedir.



Şekil 2.16. Mesaj Gönderimi Akış Şeması (Mıcrochip Inc., 2004)

2.2.4. Modülün Mesaj Gönderim Tamponları

PIC 18F458 mikrodenetleyicisi içerisinde 3 adet gönderim tamponu bulunmaktadır. Bu tamponların her biri SRAM içerisinde 14' er baytlık yer işgal eder. Eğer mikrodenetleyici mesaj tamponuna bilgi göndermek isterse, öncelikle TXREQ bitini 0 yapmalıdır. Bu işlem, gönderilmek için bekleyen mesajların gönderilebilmesini sağlamaktadır. Ayrıca TXBnSIDH, TXBnSIDL ve TXBnDLC kaydedicileri mutlaka doldurulmalıdır. Eğer mesaj çerçevesi içerisinde data baytları görünürse, o anda TXBnDm kaydedicisi de dolmalıdır. Eğer mesaj extended data çerçevesini kullanıyor ise, TXBnEIDm kaydedicisi dolmalı ve EXIDE biti 1 olmalıdır. Şekil 2.17.' da gönderim tamponunun yapısı gösterilmektedir.



Şekil 2.17. Gönderim Tampon Yapısı (Microchip Inc., 2004)

2.2.5. Modülün Mesaj Gönderim Önceliği

Gönderim önceliği, tampon içerisinde gönderilmeyi bekleyen mesajlara öncelik sırası verme işlemidir. Bu işlem CAN protokolünde 'arbitartion' işleminde kullanılan öncelik yönteminden farklıdır. Burada en yüksek öncelikli gönderim tamponu, öncelikli olarak mesajı gönderecektir. Eğer iki tampon da aynı öncelik sırasına

sahipse, en yüksek tampon numarasına sahip olan tampon mesajı gönderecektir. 4 seviyeli gönderim önceliği bulunmaktadır. Eğer herhangi bir tamponun TXP bitleri ‘11’ durumunda ise, en yüksek önceliğe sahiptir. Eğer ‘00’ ise en düşük önceliğe sahiptir.

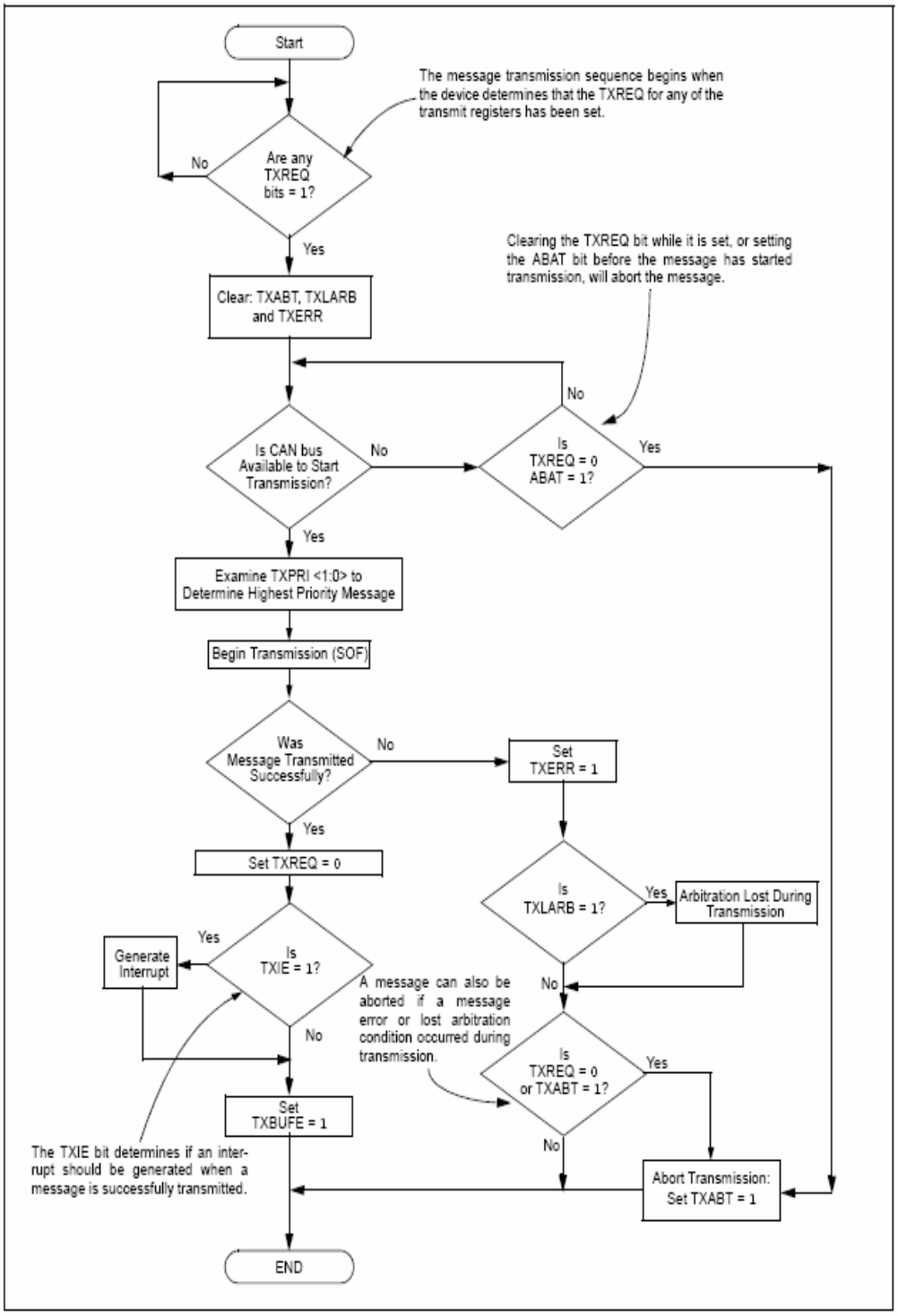
2.2.6. Modülün Mesaj Gönderimine Başlaması

Her bir tamponun mesaj akışına başlayabilmesi için TXREQ bitinin öncelikle ‘1’ yapılması gerekir. TXREQ biti ‘1’ olduğu anda, TXABT, TXLARB ve TXERR bitleri sıfırlanacaktır. TXREQ bitinin ‘1’ yapılması ile tamponlar mesaj göndermek için hazır hale gelir. Eğer modül, mesaj gönderimi için hattın uygun olduğunu tespit ederse, mesaj gönderimi başlayacaktır. Modül öncelikle en yüksek öncelik seviyesine sahip olan mesajı gönderecektir. Mesaj gönderme işlemi tamamlandıktan sonra TXREQ biti sıfırlanır. TXBnIF biti ‘1’ olur ve eğer TXBnIE biti ‘1’ olmuşsa bir kesme üretilir.

Eğer mesaj gönderiminde bir hata oluşursa, TXREQ biti ‘1’ olarak kalmaya devam eder. Bu durum, mesajın gönderim için hala tamponda beklediğini gösterir. Eğer mesaj gönderilir ve hata durumuyla karşılaşılırsa, TXERR ve IRXIF bitleri ‘1’ olur ve bir kesma üretilir. Eğer mesaj ‘arbitration’ ı kaybederse, TXLARB biti ‘1’ olur.

2.2.7 Modülün Mesaj Gönderimini Durdurması

Mikrodenetleyici, TXREQ bitini ‘0’ yaparak gönderim işleminin durmasını sağlayabilir. ABAT bitinin ‘1’ yapılması ile gönderilmek için bekleyen tüm mesajlar durdurulur. Eğer mesaj daha gönderime başlanmamışsa, veya arbitration işlemini kaybetmişse veya bir hata durumuyla karşılaşmışsa, yine durdurma işlemi gerçekleşir. ABT bitinin ‘1’ olması mesaj gönderim işleminin durdurulduğunu gösterir. Şekil 2.18.’ de mesaj gönderme işleminin akış şeması gösterilmektedir.



Şekil 2.18. Mesaj Gönderim Akış Şeması (Microchip Inc., 2004)

2.2.8 Mesaj Kabul Filtre ve Maskeleri

Mesaj kabul filtre ve maskeleri, MAB üzerinde bulunan mesajın hangi alım tamponuna yüklenmesi gerektiğine karar vermek için kullanılır. Geçerli bir mesaj MAB üzerine geldiğinde, mesajın tanımlayıcı alanları filtre değerleri ile karşılaştırılır. Eğer bir eşitlik sağlanırsa, mesaj uygun olan mesaj tamponuna yüklenir. Filtre maskeleri, tanımlayıcı bitler ile filtredeki bitlerin uygun olup olmadığını kontrol etmek için kullanılır. Çizelge 2.3.' de filtre ve maskelerin doğruluk tablosu gösterilmektedir.

Çizelge 2.3. Filtre / Maske Doğruluk Tablosu

Mask bit n	Filter bit n	Message Identifier bit n001	Accept or Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Filtrele karşılaştırma yaptığında ve mesaj alıcı tampona yüklendiğinde, mesaj kabulünü sağlayan filtrenin numarası, RXBnCON kaydedicisinin 0 ile 2. bitleri arasına (FILHIT) yazılır. RXB1CON kaydedicisi için FILHIT<2:0> bitlerinin değerleri Çizelge 2.4.' de gösterilmektedir.

Çizelge 2.4. FILHIT <2:0> Bitlerinin Değerleri

FILHIT<2:0>	**FİLTRE**
101	RXF5
100	RXF4
011	RXF3
010	RXF2
001	RXF1
000	RXF0

2.4. CAN Bus Üzerinden Bellek Alanlarının Programlanması

Güçlü bir network haberleşmesinin sağlanabilmesi için, sistem içerisinde bulunan modüllerden tek bir modülün CAN Bus üzerinden programlanması arzu edilen bir özelliktir. Doğaldır ki, bu işi yapmak da bir hayli problemli ve zordur. Çünkü modüller arasında sürekli bir mesaj akışı varken bir modülün sistem içerisinde programlanması sistemin çalışması aksatabilir (Microchip, 2003). Bilhassa hayati önem taşıyan sistemlerde bu işlemin yapılması daha da problemlidir. Bu bağlamda, bir modülün veya modüllerin programlanabilmesinde 2 farklı yol kullanılacaktır. Birincisi, modül sistem içerisinden sökülüp ayrı bir şekilde programlanması, ikincisi ise sistem üzerinden modül çıkarılmadan ve sistemin çalışması aksatılmadan programlanmasıdır.

2.4.1. Modüllerin Tek Olarak Programlanması

Çalışan bir sistem içerisindeki her modül, donanımsal olarak farklı yapılarda olabilir. Böyle bir durumda, modüllerin programlanabilmesi için, sistemin durdurulması ve sadece o modül ile iletişim sağlanarak modül programlanması gerekmektedir, veya modül sistem içerisinden çıkarılarak ayrı bir yerde programlanmalıdır (Microchip, 2003). Bahsedilen bu yöntem, modüllerin programlanmasındaki en basit yöntemdir. Çünkü, modül ile bire bir bağlantı kurulması ile programlanacak olan entegrenin bellek bölgelerine direkt erişim sağlanabilir. Fakat, bu tekniğin kullanılması için sistemin durdurulması gerekmektedir. Hayati önem taşıyan sistemlerin bu şekilde durdurulması istenmeyen bir durum oluşturur.

2.4.2. Modüllerin Sistem İçerisinde Programlanması

Çalışan bir sistemin, çalışması durdurulmadan herhangi bir modülünün programlanması çok kullanışlı bir yöntemdir. Çünkü, bir veya birden fazla modülün üzerindeki programların yenilenmesi için sistemin tamamının durdurulması gerekmez. Genellikle sistem içerisindeki CAN modüllerinin tamamı aynı donanımsal yapıya sahiptir. Aynı zamanda, CAN haberleşmesinin yapısının da mesaj tabanlı olması modüllerin programlanmasında kolaylık sağlamaktadır. Çünkü bir modülün

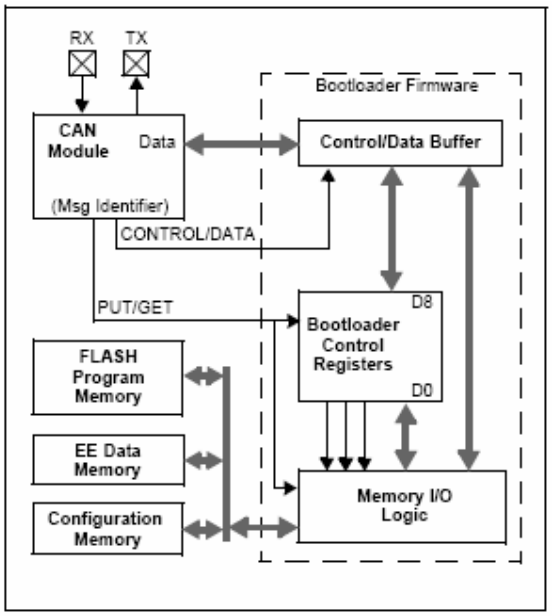
üzerine gelecek mesajın kabulü veya reddi modül üzerideki yazılım vasıtasıyla gerçekleştirilmektedir. Yani sistem içerisindeki modüllerin fiziksel adreslerine göre mesaj gönderilmediği için modülün sistem içerisinden ayrılması sistemin bütünlüğünü ve çalışmasını aksatmamaktadır.

Çalışan bir sistem içerisindeki bir modülün programlanması istenen bir durum olmakla beraber bazı önemli zorlukları da bulunmaktadır. CAN haberleşmesinde, mesajların tanımlayıcı alanlarına verilen değerler mesajın önceliğini belirlemektedir. Kritik öneme sahip sistemlerde, modüllerin programlanmasında kullanılacak olan mesajlara mı öncelik verilecek, yoksa sistemin çalışması için gerekli olan mesajlara mı öncelik verilecek bunun iyi belirlenmesi gerekir. Modül üzerine yüklenecek olan programın boyutu büyük ise ve mesaj öncelikleri modülün programlanması için kullanılacak olursa, sistemin çalışması yavaşlayabilir veya durabilir. Modülün çalışması için hayati öneme sahip mesajlara öncelik verilirse, modülün programlanması ve tekrar sistem içerisine dahil olması çok uzun sürebilir.

2.5. Microchip CANBootloader Programı

Bootloader, entegrelerin farklı iletim yolları kullanılarak programlanabilmesini sağlayan bir programdır. Bootloader programı yaygın olarak 2 çeşittir. Birincisi, entegrelerin seri portları kullanılarak programlanmasını sağlayan UART Bootloader' dır. İkincisi ise CAN Bus üzerinden programlanabilmesini sağlayan CAN Bootloader programıdır. Microchip firması da, PIC 18FXXX tipi olan ve içerisinde CAN modülü bulunan entegreleri programlayabilmek için bir bootloader programı geliştirmiştir. Yalnız bu program ile haberleşme işlemi seri port kullanılarak gerçekleştirilmektedir. Bu sistemin çalışmasında donanımsal olarak iki nokta vardır. Sistem içerisinde iki tane mikrodenetleyici bulunmaktadır. Bunlardan birincisi programlanacak olan entegreyi programlayan mikrodenetleyicidir. Her iki mikrodenetleyici üzerinde de bootloader programı yüklüdür. Birinci mikrodenetleyiciye bilgisayarın seri portundan kullanıcı programının kodları seri mesaj paketleri olarak gönderilir. Bu mikrodenetleyici gelen mesajları CAN mesajları haline çevirerek asıl programlanacak olan ikinci mikrodenetleyiciye

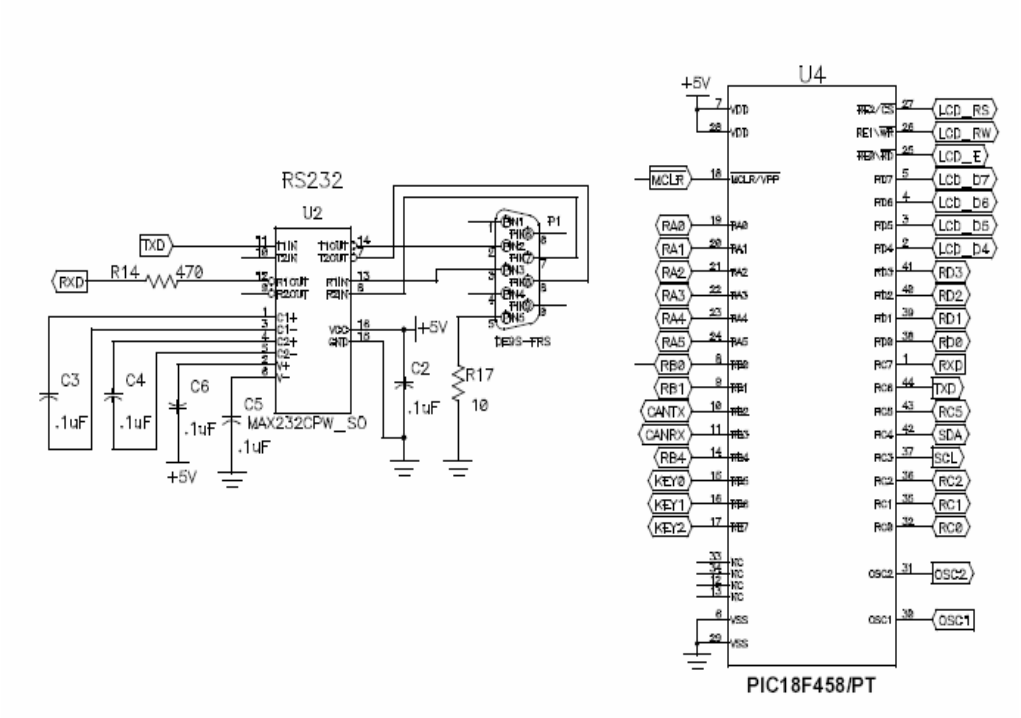
gönderir. Şekil 2.19.' da Microchip firmasının oluşturduğu bootloader programının yapısı görülmektedir.



Şekil 2.19. Microchip Bootloader Program Yapısı (Microchip, 2003)

Bootloader programı içerisine datalar, seri haberleşme uçlarından gelmektedir. Gelen bu datalar program tarafından yorumlanarak, entegrenin hangi bellek bölgesinin programlanacağı veya okunacağı belirlenir. Bu sistem içerisinde bootloader programına gelen iki farklı mesaj yapısı vardır. Bunlardan biri kontrol amacıyla gönderilen ve hangi bellek adresinin kullanılacağını gösteren mesajlardır. Diğeri ise içerisinde programlayıcı dataları bulunduran mesajlardır.

Bootloader programının donanımsal yapısı incelendiğinde entegre ile bilgisayar arasındaki haberleşme hızının 57600 Kbit/saniye olduğu görülmüştür. CAN protokolünün kullandığı haberleşme hızları düşünüldüğünde bu hızın çok yavaş olduğu görülür. Çünkü CAN protokolünün maksimum haberleşme hızı 1 Mbit/saniye' dir. Şekil 2.20.' de Microchip Bootloader programının donanımsal yapısı gösterilmiştir.



Şekil 2.20. Microchip Bootloader Donanımsal Yapısı (Microchip, 2003)

2.6. Hexadesimal Dosya Formatı

Heksadesimal dosya formatı, binary (ikili) olarak kodlanmış dataların, ASCII kod karşılıkları ile gösterilmesi için kullanılan bir yöntemdir. Heksadesimal dosyalar mikrodenetleyicilerin programlanabilmesi için uygun bir yapıya sahiptir. 8 bit uzunluğundaki bir binary data, heksadesimal dosya içerisinde 1 baytlık ASCII kodlarla gösterilmektedir. Heksadesimal dosya içerisindeki her bir satır bir kayıt bloğu anlamına gelir. Her bir kayıt ise kendi içerisinde farklı amaçlarla kullanılan parçalardan oluşur. Bir hexsadesimal kayıt bloğunun içerisinde, kayıt tipi, uzunluk, bellek adresi, data ve kontrol parçaları bulunur. Bu alanlar, mikrodenetleyicinin programlanabilmesinde programcı açısından hayati öneme sahiptir (Intel Corporation, 1998). Kayıt bloğu içerisinde bulunan kayıt tipi alanı içerisindeki sayılara göre 6 farklı kayıt bloğu bulunur. Bunlar;

- Data Kaydı
- Dosya Sonu Kaydı
- Genişletilmiş Segment Adres Kaydı
- Başlangıç Segment Adres Kaydı
- Genişletilmiş Lineer Adres Kaydı
- Başlangıç Lineer Adres Kaydı

Bu kayıt tipleri, kullanılan mikrodenetleyicinin yapısına göre kullanım farklılıkları gösterir. Mikredenetleyici 8 bitlik yapıya sahip ise Data ve Dosya Sonu kayıt tipleri kullanılır diğerleri kullanılmaz. Diğer kayıt tipleri ise 16 veya 32 bitlik mikrodenetleyicilerin yapısına uygundur. Şekil 2.21.' de Heksadesimal Dosya Formatı gösterilmektedir.

RECORD MARK ':'	RECLEN	LOAD OFFSET	RECTYP	INFO or DATA	CHKSUM
1-byte	1-byte	2-bytes	1-byte	n-bytes	1-byte

Şekil 2.21. Heksadesmal Dosya Formatı

Her bir kayıt bloğu 1 baytlık ":" işareti ile başlar. Bu işaret kayıt başı anlamına gelir. Aynı zamanda, bu alan, programlayıcı açısından kayıt bloklarının birbirlerinden ayrılması için kullanılabilir. Bu 1 baytlık alanın arkasından, kayıt bloğu içerisindeki data alanında kaç baytlık data olduğunu gösteren kayıt uzunluk alanı gelir. Bu alan da blok içerisinde 1 baytlık yer işgal eder. Kayıt bloğunun bir sonraki alanı olan ofset alanı, data alanı içerisindeki dataların mikrodenetleyici içerisinde hangi bellek adresinden itibaren kaydedileceğini gösterir. Bu alan bellek içerisinde 2 baytlık yer işgal eder. Bu alandan sonra gelen alan ise, kayıt bloğunun tipini belirler. Çizelge 2.5.' de kayıt tipi alanının içeriği gösterilmektedir.

Çizelge 2.5. Kayıt Tipi Alan İçeriği

Kayıt Tipi	Anlamı
00	Data Kaydı
01	Dosya Sonu Kaydı
02	Genişletilmiş Segment Adres Kaydı
03	Başlangıç Segment Adres Kaydı
04	Genişletilmiş Lineer Adres Kaydı
05	Başlangıç Lineer Adres Kaydı

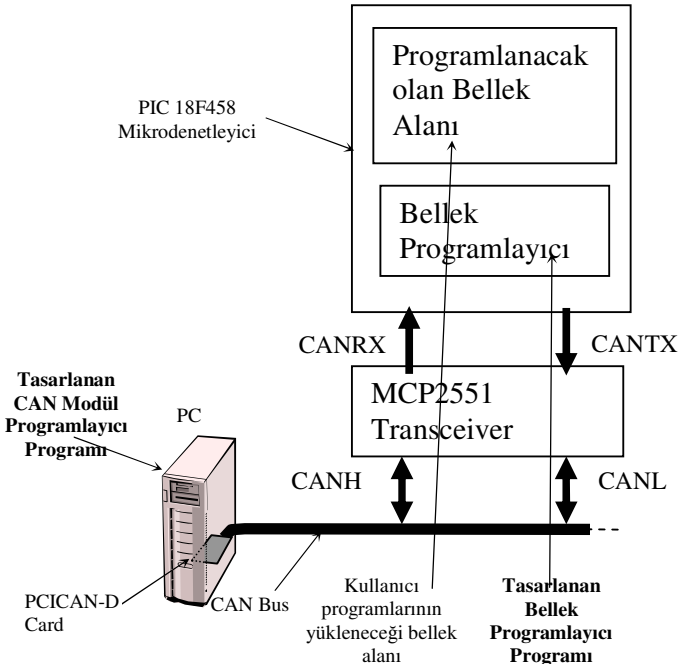
Data alanı içerisinde, mikrodenetleyici içerisine kaydedilecek olan datalar bulunur. En son alan olan kontrol alanı ise, bazı özel alanlar içerisindeki dataların 2' ye göre komplementlerinin toplamlarını içerisinde barındırır (Intel Corparation, 1998). Bu alan ile, kayıt bloğunda bir uyuşmazlığın olup olmadığı anlaşılır.

3. MATERYAL VE METOD

3.1. CAN Modül Programlayıcısının Donanımsal Yapısı

Bilgisayar üzerinden bir PIC mikrodenetleyicisinin CAN iletişim kuralları kullanılarak programlanabilmesi için iki önemli noktanın oluşturulması gerekmektedir. Birincisi donanım olarak elektronik devrenin tasarlanması, ikincisi ise elektronik devrenin yönetilmesi ve mikrodenetleyicinin programlanması için gerekli programların hazırlanmasıdır. Öncelikle mikro denetleyiciyi programlayacak olan sistemin donanımsal devre yapısı oluşturulmuştur.

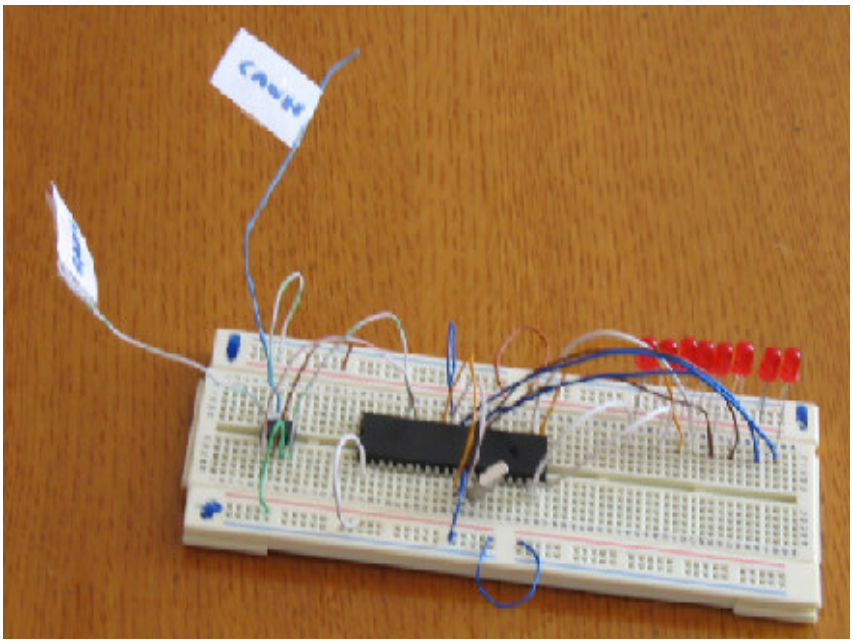
Elektronik devre yapısı içerisinde bir PIC mikro denetleyici bulunmaktadır. Bu mikrodenetleyici içerisinde CAN haberleşmesini sağlayan modül bulunur. Bu modül yardımı ile bilgisayar üzerinden CAN kablosu ile haberleşmek mümkündür. Aynı zamanda bu mikro denetleyici PIC uygulamalarının çalıştırabileceği portlara sahiptir. Bu portlara bağlı display, robot kol gibi donanımsal parçalar prototip olarak sistem içerisine dahil edilebilir. Sistemin çalıştırabilmesi için PIC mikro denetleyicinin program belleğine uygun program yerleştirilmesi gerekmektedir. Bunun gerçekleştirilebilmesi, yazılacak olan programlar ile mümkün olabilir. Şekil 3.1.' de uygulamanın donanımsal yapısı gösterilmiştir.



Şekil 3.1. Uygulamanın Donanımsal Yapısı

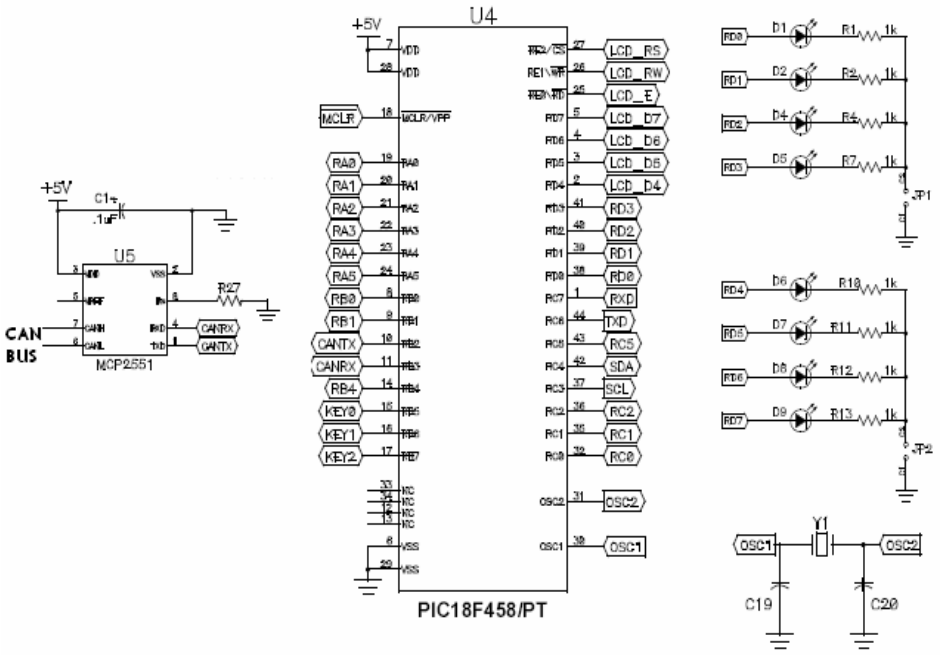
Sistem içerisinde donanımsal yapıyı oluşturan 3 temel parça bulunur. Bunlar PCI CAN Kartı, CAN Bus Kablosu ve PIC tabanlı elektronik CAN devresidir. PCI CAN Kartının kendisi de bir CAN Modülü olarak çalışmaktadır. Bu kart üzerinden mikrodenetleyiciyi programlayacak olan datalar CAN Bus üzerine yüklenir. CAN Bus kablosu UTP Katagori 5 tipi kablo olup iki ucunda DB9 dişi seri port konnektörü bulunur. Bu kablo üzerinde CANH ve CANL uçları arası 120’ şer ohm ile sonlandırılmıştır (Şekil 3.2). Sistem içerisinde kullanılan PIC tabanlı elektronik CAN devre görüntüsü Şekil 3.3.’ de verilmiştir.

Şekil 3.2. CAN Kablosu



Şekil 3.3. Uygulama Devresi

Uygulama devresinde, iki temel malzeme kullanılmaktadır. Bunlardan biri, programlanacak olan PIC 18F458, diğeri ise CAN modülünün CAN Bus üzerine çıkmasını sağlayan MCP2551 Transceiver' dır. CAN Bus üzerinden gelen mesajlar transceiver üzerinde, mikrodenetleyicinin kabul edeceği bir voltaj seviyesine çevrilir. Transceiver' in çıkış uçları olan TXD ve RXD uçları mikrodenetleyicinin CANTX ve CANRX uçlarına bağlanır. Bu sayede CAN Bus üzerinden gelen mesajlar mikrodenetleyici tarafından kabul edilir. Şekil 3.4.' de uygulama devre şeması gösterilmektedir.

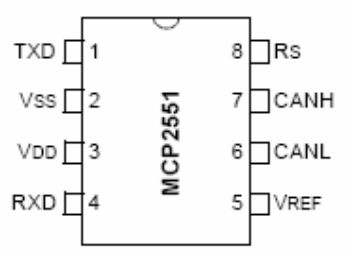


Şekil 3.4. Uygulama Devre Şeması

Uygulama devresinde kullanılan kristal osilatör 4 MHz. Olarak seçilmiştir. Mikrodenetleyici üzerinde çalışacak programın yapısına göre bu osilatörün değeri değiştirilebilir. MCP2551 Transceiver entegresinin CANL ve CANH çıkışları CAN Kablosunun sırasıyla 2 ve 7. uçlarına bağlanır. Bu sayede sistem CAN Bus üzerine bağlanmış olur. Elektronik devrenin oluşturulmasından sonra mikrodenetleyicinin programlanabilmesi için 2 tane program yazılmıştır. Bunlar, CAN Modül Programlayıcı programı ve Bellek Programlayıcı programıdır.

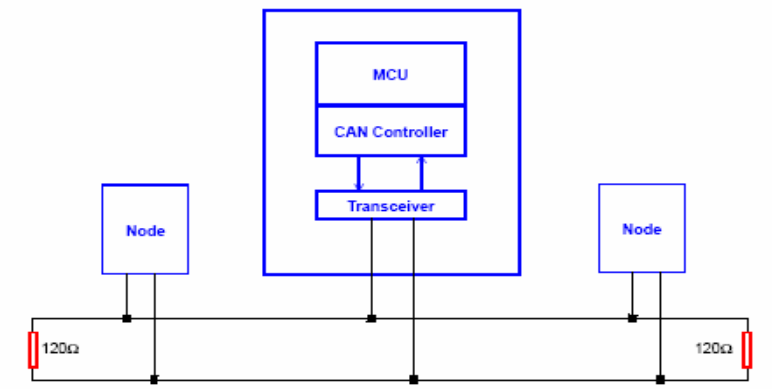
3.2. MCP2551 Transceiver

MCP2551 entegresi, CAN Bus üzerinden gelen mesajları mikrodenetleyicinin CAN modülüne, CAN modülünden gelen mesajları CAN Bus' a aktaran yüksek hızlı CAN alıcı vericisidir. Şekil 3.5.' de entegrenin bacak bağlantıları gösterilmektedir (Microchip Inc., 2004).



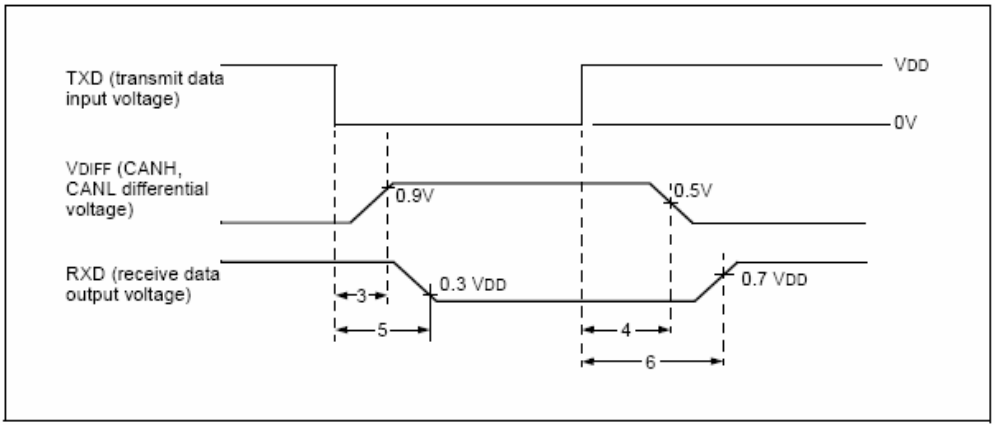
Şekil 3.5. MCP2551 Entegre Bacak Bağlantıları

CAN haberleşme sistemi içerisinde bulunan her bir modül, CAN Bus kablosu üzerine yükleyeceği dijital sinyalleri, iletim için uygun bir seviyeye çekmeleri gerekir. MCP2551 entegresi, CAN modülü ile bus üzerinde dış kaynaklardan dolayı oluşan yüksek gerilim dalgalarını tamponlama özelliğine sahiptir. Şekil 3.6.' da mikrodenetleyicinin CAN bus üzerine bağlantısı gösterilmektedir.



Şekil 3.6. Mikrodenetleyicinin CAN Bus Üzerine Bağlantısı

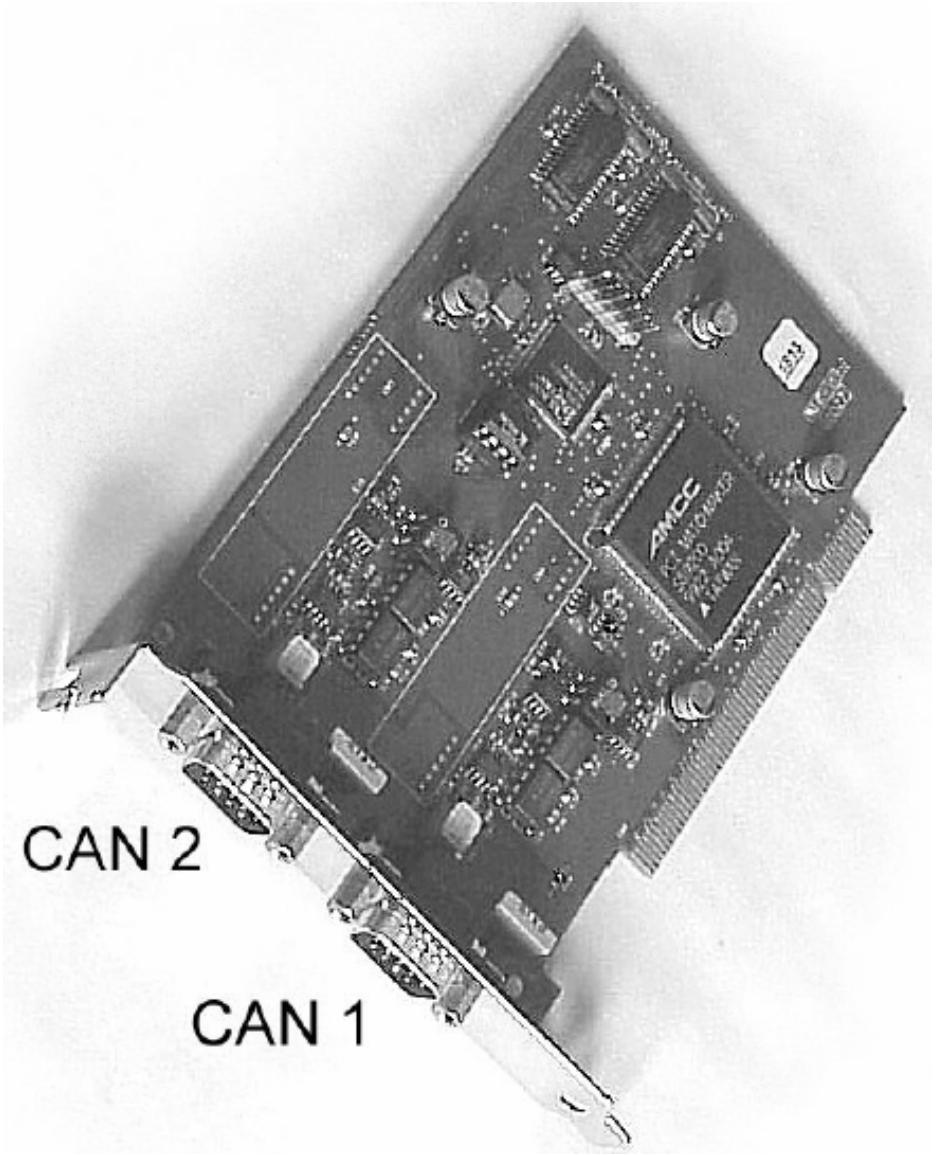
CAN Bus üzerinde 2 tane lojik seviye bulunur. Bunlar Dominant (lojik 0) ve Recessive (lojik 1) gerilim seviyeleridir. Dominant lojik seviye, CANH ve CANL arasındaki diferansiyel gerilim tanımlı olan gerilim seviyesinin üzerinde olduğu zaman oluşur (Microchip Inc., 2004). Tanımlı olan seviye, 1.2V civarındadır. Recessive lojik durum ise, iki hat arasındaki diferansiyel gerilimin tanımlı olan seviyenin altında olduğu zaman oluşur. Genelde bu seviye 0V' tur. Şekil 3.7.'de CANH ve CANL arasındaki gerilim seviyeleri gösterilmektedir.



Şekil 3.7. CAN Bus Lojik Gerilim Seviyeleri (Microchip Inc., 2003)

3.3. PCI CAN Kartı

Kvaser firması tarafından üretilen PCIcan-HS/HS CAN Kartı, CAN Modüllü sistemler ile bilgisayarları birbirlerine bağlamak için tasarlanmıştır.Kartın çıkışında 2 tane CAN Portu vardır. Kart, iki ayrı CAN modülünü aynı anda bilgisayar ile haberleştirebilmektedir. Şekil 3.8.' de PCI CAN-HS/HS kartı gösterilmektedir.



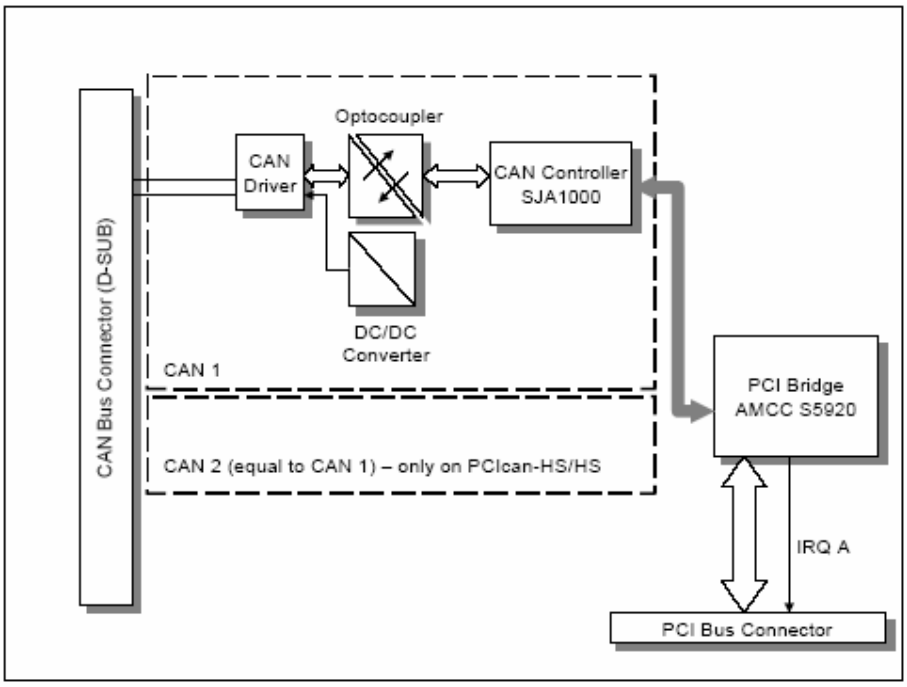
Şekil 3.8. PCIcan-HS/HS CAN Kartı (Kvaser AB, 2002)

Kendiside bir nevi CAN Modülü olan kart, bilgisayarın PCI slotuna takılır. Kartın çıkışında iki farklı CAN sistemine bağlanmaya olanak tanıyan 9 pinli DB9 konnektör bulunur. İçerisinde bus sürücü devresi için 82C251 barındırır. Fiziksel katmanı ISO 11898 ile standartlaştırılmıştır. Çalışma hızı maksimum 1 Mbit / saniye' dir. Kartın genel özellikleri Çizelge 3.1' de verilmiştir.

Çizelge 3.1. PCI CAN Kart Genel Özellikleri (Kvaser AB, 2002)

General	
Size	PCI-bus PC card (135 mm)
Power consumption	PCIcan: max 600 mA @ 5V.
CAN bus connector	9-pin DSUB, male, for PCIcan-HS CAN bus. 2 x 9-pin DSUB, male, for PCIcan-HS CAN buses.
CAN Controller(s)	
PCIcan-HS : 1 x SJA1000 PCIcan-HS/HS : 2 x SJA1000	
CAN Clock frequency: 16 MHz	
CAN Bus Driver(s)	
Drivers	PCIcan-HS, -HS/HS: Philips 82C251; compliant with the ISO 11898 standard.
Voltage feed	The drivers are galvanically separated (selectable by switches) from the power supply on the PC by on-board DC/DC converters.
Grounding	The ground of the CAN drivers is available at the DSUB connector.
Other Features	
• Fast optocouplers between CAN circuits and drivers. • CAN driver part fed by the PC through DC/DC-converters	

PCI CAN Kart içerisinde, CAN Bus üzerine bağlantı kurulabilmesi için CAN sürücüsü, SJA1000 tipi bir CAN Kontrolcüsü bulunmaktadır. Aynı zamanda PCI Bus üzerine bağlanabilmesi için PCI Bridge denetleyicisi bulunmaktadır. Şekil 3.9.' da kartın iç yapısı gösterilmiştir.



Şekil 3.9. PCI CAN Kartının İç Yapısı (Kvaser AB, 2002)

Kart üzerinde bulunan iki tane CAN Port çıkışlarının 2 ve 7 numaralı uçları CAN Bus üzerine girmek için kullanılır. Çizelge 3.2' da CAN Kartının çıkış uçları tanımlanmıştır.

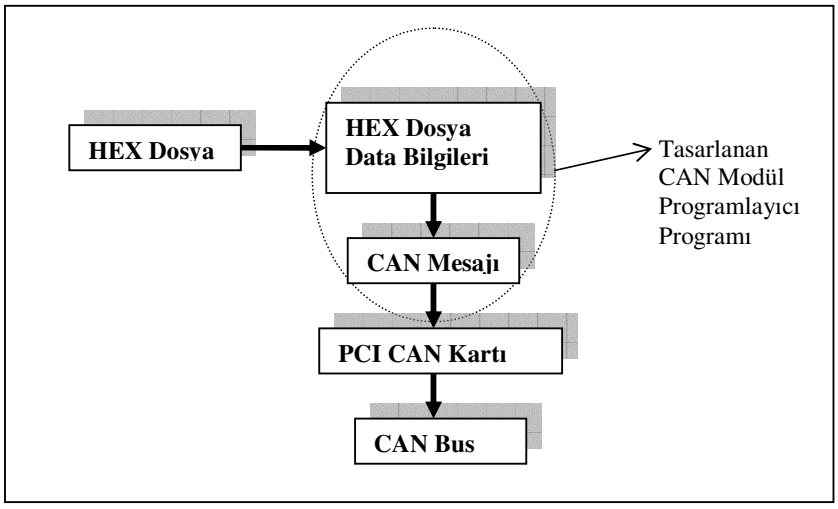
Çizelge 3.2. PCI CAN Kart Port Çıkış Uçları

Pin	Function
2	CAN-L
7	CAN-H
3	Signal ground.

Kartın sürücüleri, Windows 95/98/ME ve Windows NT/2000/XP işletim sistemlerini desteklemektedir.

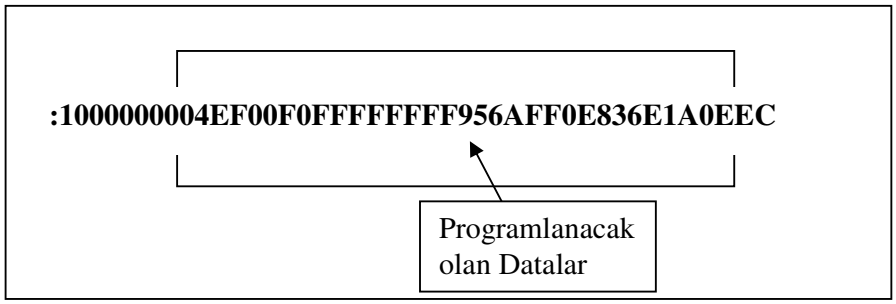
3.4. CAN Modül Programlayıcı Programı

PC üzerinde çalışması ve mikrodenetleyinin, bu program vasıtasıyla gönderilen anlamlı CAN Mesajlarına göre programlanabilmesi amacıyla tasarlanmıştır. Bu program, kullanıcı tarafından hazırlanmış olan uygulama heksadesimal dosyası içerisindeki kodları alarak, uygun CAN Mesajları haline getirir. Uygulama heksadesimal dosyası EK – 4' de verilmiştir. Bu mesajlar, mikrodenetleyici üzerinde, daha önceden paralel bir programlayıcı tarafından üzerine yüklenmiş olan bellek programlayıcı programı ile karşılanır. Şekil 3.10.' da Program çalışmasının blok şeması gösterilmiştir.



Şekil 3.10. Program Blok Şeması

Heksadesimal dosya içerisinde bulunan ve programlamaya esas teşkil edecek olan her bir kayıt bloğu içerisindeki datalar, 1' er bayt olarak alınarak text özelliğe sahip bir dosya içerisine aktarılır. EK – 5' de text dosya içeriği verilmiştir. Şekil 3.11.' de kullanıcı uygulama programının heksadesimal formu içerisindeki 1 kayıt bloğu ve bu kayıt bloğu içerisinde kod belleğe yüklenecek olan datalar gösterilmiştir.



Şekil 3.11. Uygulama Dosyası Kayıt Bloğu

Heksadesimal bir kayıt bloğu içerisindeki her bir değer 4 bitliktir. Programlanacak olan data bloğunda toplam 32 değer vardır. 1 bayt 8 bitten oluşmaktadır. Her iki heksadesimal sayı 1 bayt eder. Yani kayıt bloğu içerisinde programlanacak olan toplam 16 baytlık data vardır. Data alanı içerisindeki her bayt CAN Modül programlayıcı program vasıtasıyla desimal sayıya çevrilerek text bir dosyaya aktarılır. Şekil 3.12' de text dosya içeriği gösterilmiştir.

4, 239, 0, 240, 255, 255, 255, 255, 149, 106, 255, 14, 131, 110, 26, 14

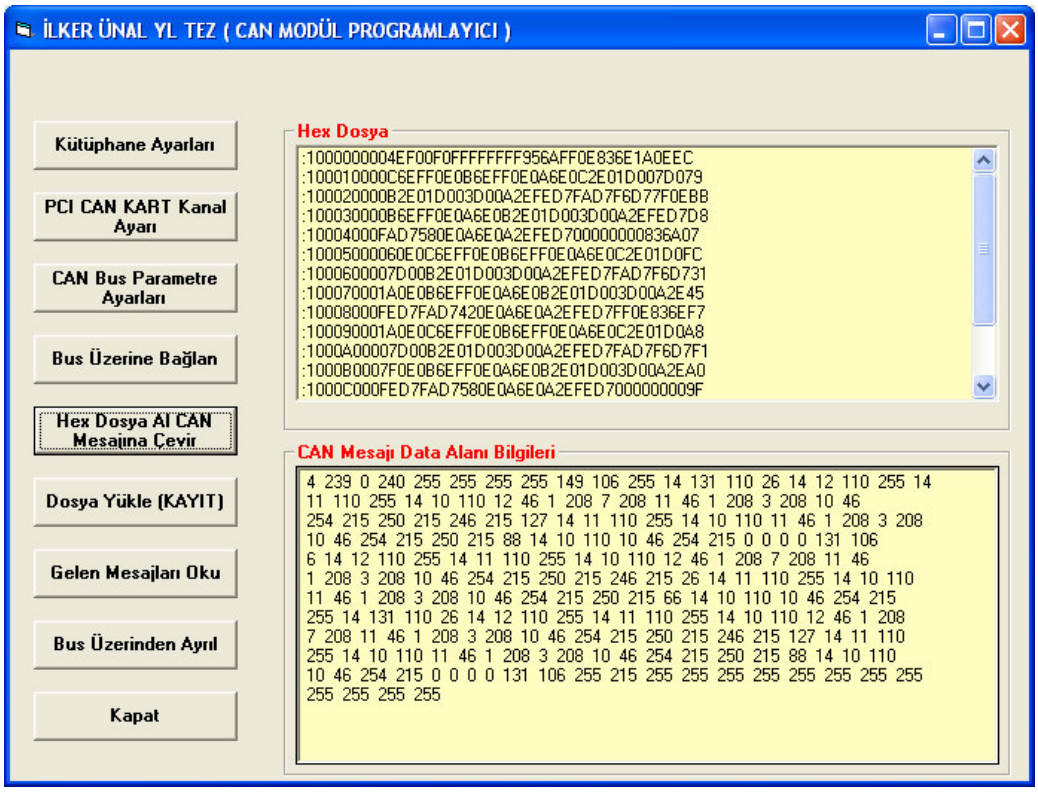
Şekil 3.12. Heksadesimal 1 kayıt Bloğunun Text Dosya içeriği

Text dosya içerisinde bulunan her 1 baytlık desimal data, CAN mesaj ID' si '01' olacak şekilde mesaj içerisine yüklenerek PCI CAN Kartı üzerinden CAN Bus üzerine aktarılır. Şekil 3.13.' de uygulama programı sonucunda oluşan CAN Mesajının yapısı gösterilmiştir.

Mesaj ID	**Uzunluk**	**Data**
00000000001	0001	Text Dosyadan alınan 1 baytlık data

Şekil 3.13. Program Data CAN Mesaj Formatı

CAN mesaj çerçevesi içerisinden gönderilen ve ID' si '01' olan mesajların gönderiminin bitmesi sonucunda, mikrodenetleyici içerisine yüklenen uygulama programının çalıştırılması amacı ile ID' si 02 olan bir mesaj gönderilir. Şekil 3.14.' de PC üzerinde çalışan CAN Modül Programlayıcı programının görüntüsü verilmiştir.



Şekil 3.14. CAN Modül Programlayıcı Görüntüsü

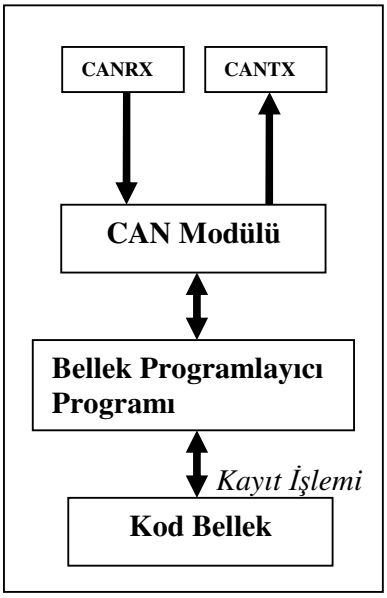
Program Visual Basic programlama dili kullanılarak yazılmıştır. Program üzerinde kullanılan butonların her birinin, programın çalışabilirliliği açısından farklı amaçları mevcuttur. Kullanılan PCI Can Kartının programa tanıtılması için kütüphane ayarlarının yapılması gerekir. PCI CAN Kartının çıkışında 2 adet CAN Portu bulunur. Bu portlardan hangisinin kullanılacağı kanal ayarlaması yapılarak tespit edilir. Programlanacak olan CAN Modülü ile bilgisayar arasındaki iletim hızı CAN Bus parametre ayarları yapılarak belirlenir. İki sistem arasında iletişimin başlayabilmesi için Bus üzerine bağlanılması gerekir. Bu aşamadan sonra, kullanıcı

tarafından hazırlanan Heksadesimal dosya seçilerek, mikrodenetleyiciyi programlayacak olan datalar, CAN Bus üzerinden mikrodentleyiciye gönderilir. Kullanıcı istediği zaman karşı sistemden gelen mesajları da okuyabilir. Program sadece mikrodenetleyicinin programlanmasında kullanılması yanında mikrodenetleyici tarafından gelen mesajları görüntülemektedir. İstenildiğinde program üzerinde yapılacak değişikliklerle farklı programlar geliştirilebilir. Sonuç olarak gelen mesajlar farklı programlarda kullanılabilir.

Program, her 1 saniyede bir programlanacak olan dataları CAN Bus üzerine yüklemektedir. Burada kullanılan 1 saniyelik aralık, mikrodenetleyici içerisinde çalışan bellek programlayıcı programı ile senkronizasyonu sağlar. Bellek programlayıcı programı da 1 saniyede bir gelen mesajları alarak işler. Burada verilen süre kullanıcıya göre artırılabilir veya azaltılabilir. Bu programda, EK - 3' de verilen kullanıcı uygulama programı kullanılmıştır. Kullanıcı uygulama programı, 1 saniyelik ve 5 saniyelik aralıklarla mikrodenetleyicinin D portuna takılan 8 adet led grubunu yakıp söndürmektedir. Kullanıcı uygulama programı değiştirilerek farklı programların yüklenmesi sonucunda çalıştırılması da sağlanabilir. Programın yazımında Kvaser firmasının ürünü olan PCI CAN Kartının Visual Basic kütüphanesi kullanılmıştır. Program farklı firmaların ürettikleri CAN Kartlarına da destek vermektedir. PC üzerinde çalışan CAN Bus programlayıcısı ile mikrodenetleyici arasındaki iletişim hızı 250 KBit/saniye' dir. Bu hız, programlar içerisinde yapılacak küçük değişikliklerle 1 MBit/saniye' ye kadar çıkarılabilir. CAN Modül programlayıcısına ait Visual Basic kodları EK - 1' de verilmiştir.

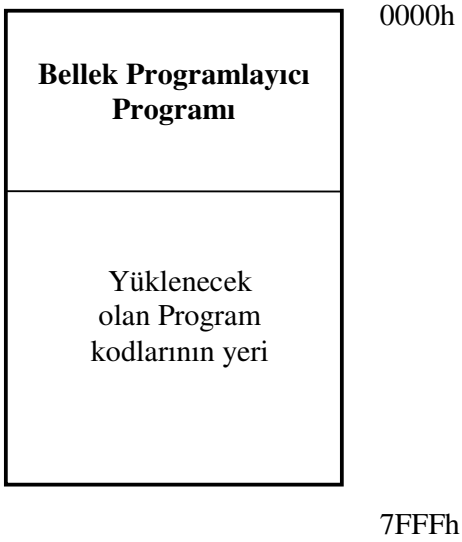
3.5. Bellek Programlayıcı Programı

Bellek programlayıcı programı, PC üzerinde çalışan ve CAN Bus programlayıcısı programından gelen mesajları, ID' lerine göre yorumlayarak işlem yapan bir yapıya sahiptir. Şekil 3.15.' de bellek programlayıcı programının yapısı gösterilmiştir.



Şekil 3.15. Bellek Programlayıcı Programının Yapısı

Bellek programlayıcı programı, öncelikli olarak paralel bir programlayıcı kullanılarak mikrodenetleyici içerisini yüklenir. Bellek programlayıcı programının kod bellek içerisindeki yapısı Şekil 3.16' da gösterilmiştir.



Şekil 3.16. Bellek Programlayıcı Programının Bellek Yapısı

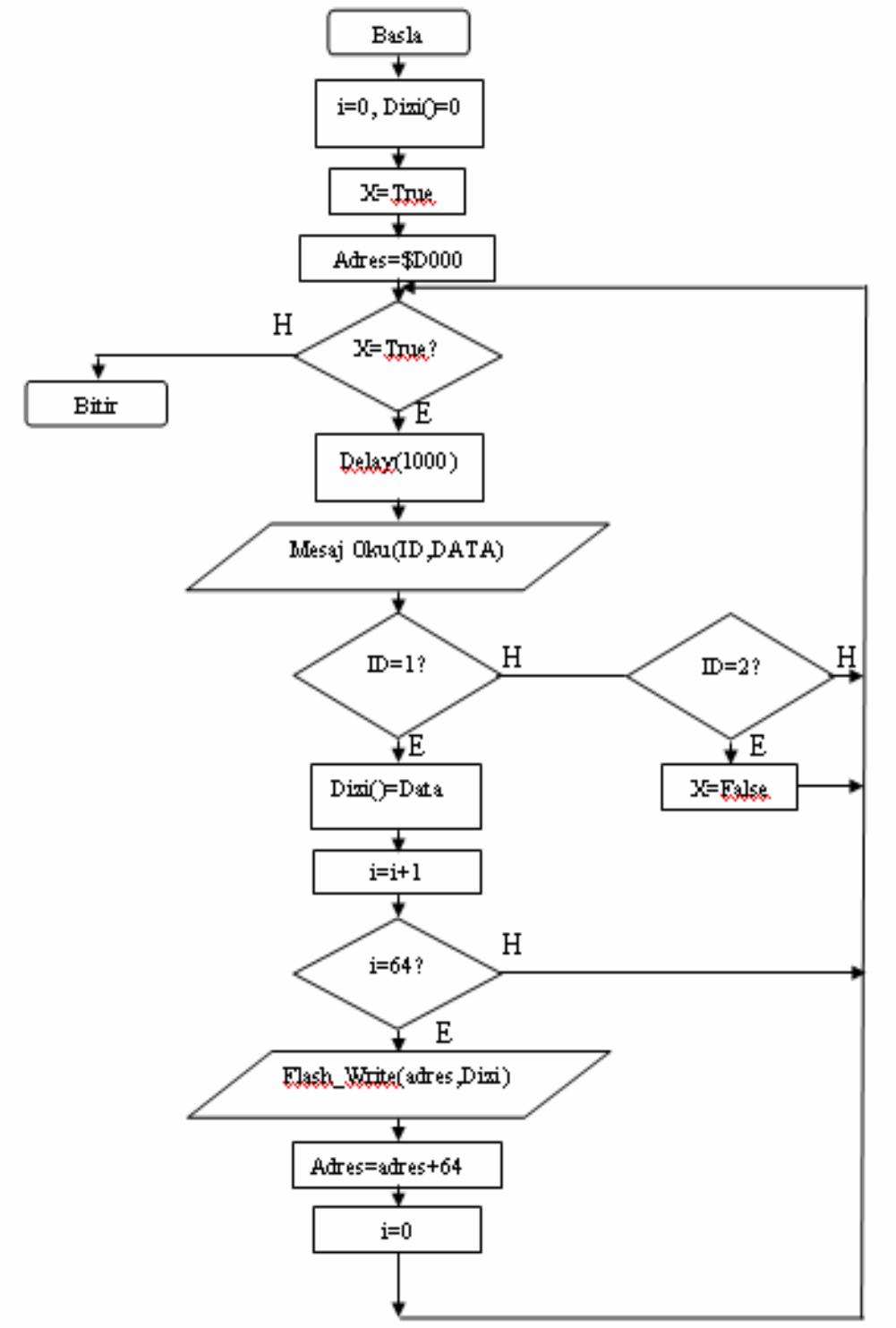
Bellek Programlayıcı programı, kod belleğin 0000h adresinden itibaren yüklenir. Mikrodenetleyici içerisine yüklenmiş olan program, PC tarfından gönderilen CAN Mesajlarının yapısına göre hareket eder. Gelen mesajın ID alanında bulunan değer, gelen mesajın durumunu ve bu mesaj karşısında bellek programlayıcı programının nasıl bir yol izleyeceğini belirler. Çizelge 3.3.' de gelen mesajın ID' sine göre programın izleyeceği yol gösterilmiştir.

Çizelge 3.3. Gelen Mesaj ID İşlemleri

Gelen Mesaj ID	**İşlem**
01	Programlanacak olan data
02	Programın çalıştırılması

CAN Bus üzerinden okunan ve mesaj ID' si '01' olan her mesaj bellek programlayıcı programı tarafından programlanacak data olarak algılanır. Her gelen mesaj içerisinde 1 baytlık data bulunur. Gelen bu mesajlar içerisindeki her 1 baytlık data, program tarafından bir dizi içerisine yüklenir. ID değeri '01' olan mesajların alımı tamamlandıktan sonra, '02' ID numaralı mesaj gelir. Her 64 baytta bir dizi içerisine yüklenen datalar, kod bellek içerisinde, belirlenen bir bellek adresinden itibaren kaydedilir. Programlama işlemi bittikten sonra CAN bus üzerinden gelen ve ID' si '02' olan mesaj ile yüklenen program çalıştırılır. Bellek Programlayıcı programının kodları EK – 2' de verilmiştir.

Bellek programlayıcı programı, Mikroelektronika firmasının ürünü olan MikroBasic Programı kullanılarak yazılmıştır. MikroBasic Programı PIC tipi mikrodenetleyiciler için kullanıcı uygulama programları geliştirmek için kullanılmaktadır. Şekil 3.17.' de bellek programlayıcı programının akış şeması verilmiştir.



Şekil 3.17. Bellek Programlayıcı Programı Akış Şeması

4. ARAŞTIRMA BULGULARI

CAN Protokolü kullanarak kurulan bir haberleşme ağında, sistem içerisinde çalışan modüllerin zaman zaman üzerlerinde çalışan programları yenilemeleri gerekebilir. Çoğu network sisteminde, programlanacak olan modüller sistem içerisinden sökülerek programlanabilmektedir. Bu durum, sistemin çalışmasını yavaşlattığı gibi, hayati önem taşıyan işlemler de sistem içerisinde çalışamaz hale getirir. Bu problemlerin çözümü için, modülün, sistem içerisinden sökülmeden programlanması gerekir.

Çalışmanın ikinci bölümünde anlatılmış olan CAN protokolünün genel yapısı incelendiğinde, bir modülün, bu yapıyı kullanarak CAN hattı üzerinden programlanabileceği kanısına varılmıştır. Çünkü, programlanacak olan datalar, mikrodenetleyici üzerine mesajlar halinde gönderilmektedir. Çoğu mikrodenetleyicinin programlanması için yapılan iki önemli nokta vardır. Birincisi, programlanacak olan dataların entegrenin hangi bellek adresine yazılacağıdır. İkincisi ise bu bellek adresine gönderilecek olan datalardır. Her iki bilgi de, bilgisayar üzerinde çalışan bir program vasıtasıyla entegre üzerine gönderilmektedir. Eğer entegre üzerinde gelen bu dataları yorumlayacak olan bootloader tarzında bir program mikrodenetleyici içerisinde yüklü ise entegre rahatlıkla programlanabilir. Çalışmanın üçüncü bölümünde bu işlemler anlatılmıştır.

CAN protokolü kullanılarak kurulan haberleşme ortamları genellikle güçlü bir veri iletişiminin istenildiği yerlerdir. Bir otomobil içerisindeki güvenlik sistemleri veya bir fabrikadaki üretim sistemleri gibi hayati önem taşıyan alanlarda kullanılmaktadır. Bu ortamlarda dataların modüller arasında kesintisiz bir şekilde gönderilmesi gerekmektedir. Durum böyle olunca sistemlerin hızlı bir şekilde programlanması gerekir. Bu çalışmalar farklı entegre üretici firmalar tarafından yapılmaktadır. Yapılan çalışmaların çoğu, üretici firmalar tarafından geliştirilen bootloader programına göre yapılmaktadır. Bu çalışmada ise, kullanıcı kendi yazmış olduğu bootloader tarzında bir program ile de entegreyi programlayabileceği gösterilmiştir.

5. SONUÇ

Günümüzde çoğu CAN tabanlı entegre üretici firmalar, üretmiş oldukları entegrelerin UART veya CAN Bus üzerinden programlanabilmeleri için bootloader programları oluşturmaktadırlar. Bootloader programı, entegre içerisine yerleştirilen ve amacı kullanıcı programlarını entegreye yüklemek olan bir programdır. Bu program, entegrenin kod bellek kısmında, kullanıcı programlarından ayrı bir bellek bölgesine yüklenmektedir. Fakat üretilen bootloader programları, istenilen gibi çalışmamaktadır. Aynı zamanda bu programlar, anlaşılamaz bir yapıya sahiptir. Bu sebeple, kullanıcılar kendi bootloader programlarını kendileri yazmaktadır. Bu programlar, kullanıcılar tarafından farklı şekillerde de yazılabilir.

CAN protokolü kullanılarak, CAN modüllerinin, bilgisayar yardımı ile programlama ve çalışmalarını izleme işlemleri endüstride uygulanmaktadır. Kullanılan denetleyiciler genellikle intel tabanlıdır. Microchip firmasının üretmiş olduğu PIC tabanlı işlemcilerin CAN üzerinden programlanması ve gerekli yazılımların hazırlanması henüz tamamlanmamıştır. Bu çalışma ile PIC denetleyicilerin CAN üzerinden programlanması sağlanarak endüstrideki bu eksiklik giderilmiş olacaktır.

Bu çalışmada, kullanıcılar tarafından yazılan bir uygulama programının, CAN Bus üzerinden bir PIC mikrodenetleyicinin içerisine nasıl programlanılacağı anlatılmıştır. Mikrodenetleyici içerisine yüklenecek dosyalar heksadesimal formattadır. Bu format içerisinde, bellek içerisine kaydedilecek olan data alanları okunarak, CAN mesajları halinde mikrodenetleyiciye gönderilmiştir. Mikrodenetleyiciye gelen bu mesajların içerisindeki data alanları ile belleğin, ilgili bellek alanları programlanmıştır. Uygulama ile ilgili iki tane program yazılmıştır. Bu programlardan birincisi, programlanacak olan dataları üreten CAN Modül Programlayıcı programıdır. Diğeri ise bellek alanlarını programlayan Bellek Programlayıcısı programıdır. Yazılan bu programlar incelendiğinde, her kullanıcının kendine göre bir bellek programlayıcı programı yazabileceği gösterilmiştir.

KAYNAKLAR

Bosch, 2006. http://www.semiconductors.bosch.de/en/30/5-networking.asp

Ekiz, H., Kutlu, A., Baba, M.D., 1996. Performance Analysis of a CAN / CAN Bridge, International Conference on Network Protocols (ICNP '96), 181-188.

Farsi, M., Ratcliff, K., Barbosa., 1999. An Overwiew of Control Area Network, IEE Computing and Control Engineering Journal, 10, 3, 113-120.

Fosler, R.M., 2003. A Can Bootloader for PIC18F CAN Microcontrollers, Microchip Technology Inc., U.S.A.

Gaujal, B., Navet, N., 2005. Fault Confinement Mechanisms on CAN: Analysis and Improvements, IEE Transactions On Vehicular Technology, 54, 3, 1103-113.

Hopkins, A., 2003. Controller Area Networks, Computer Science Seminar Spring, Minnesota.

Intel Corporation , 1998. Intel Hexadecimal Object File Format Specifications.

Kavlico Corp., 2000. http://sensorsmag.com/articles/1000/18/main.shtml (1/01/2000)

Kutlu, A., 2004. Microlab: A Web based Multi - User Remote Microcontroller Labaratory for Engineering Education, Int. J. Engng. Ed., 20, 5, 879-885.

Kvaser AB, 2002. PCIcan-HS and HS/HS Datasheet, Kvaser Corporation, Sweden.

Lawrenz, W., 1997. CAN System Engineering : From Theory to Practical Applications, Springer-Verlag, New York.

Microchip Inc., 2003. High Speed CAN Transceiver Datasheet, Microchip Technology Inc., U.S.A.

Microchip Inc., 2004. PIC 18FXX8 Datasheet, Microchip Technology Inc., U.S.A.

Microchip, 2003. A CAN Bootloader for PIC18F Microcontrollers, Microchip Technology Inc., U.S.A.

Nolte, T., Hansson, H., Norström, C., Punnekkat, S., 2001. Using Bit - Stuffing Distributions in CAN Analysis, IEE Real Time Embedded System Workshop.

OSI Model, 2004. http://homepages.uel.ac.uk/u0115449/myweb14/OSI %20Model .htm (08/04/2004)

Pazul, K, 1999. Controller Area Network (CAN) Basics, Microchip Technology Inc., U.S.A.

Richards, P., 2002. A CAN Physical Layer Discussion, Microchip Technology Inc., U.S.A

Richardson, P., Seih, L., Haniak, P., 2001. A Real Time Control Network Protocol or Embeded Systems Using Controller Area Network (CAN), IEEE Elec. And Inf. Tech. Conference.

Tindell, K., Burns, A., Wellings, A., 1995. Calculating Controller Area Network, (CAN) Message Response Time, Control Engineering Practice, 3, 8, 1163-169.

Tindell, K.W., Wellings, A.J.,1994. Analysing Real - Time Communications : Controller Area Network (CAN), Proc. Real-Time Systems Syposium, 259-263.

Wei, C.L., Cullen, J.D., Al-Shamma'a, A.I., 2005. Real Time Can Network System For Monitoring The Vibrations of Vehicle Engines, Conference The Institute of Physics, 15 s.

Zuberi, K.M., Shin, K., 1995. Non – Preemptive Scheduling of Messages on Control Area Network for Real – Time Applications, In Proc. Of the IEEE Real Time Applications Symposium, 240 – 249.

EKLER

EK – 1

CAN Modül Programlayıcı Programı Kodları

Modül

```
Rem
Rem                Copyright 1998 by KVASER AB, Mölndal, Sweden
Rem                    WWW: http://www.kvaser.se
Rem
Rem  ---------------------------------------------------------------------
Rem
Rem This file contains definitions for 32-bit CANLIB32.DLL only.
Rem You have to use VB4 or later version.
Rem
Option Explicit

'
' canstat.h
'

' Don't forget to update canGetErrorText in canlib.c!
Public Const canOK = 0
Public Const canERR_PARAM = -1          ' Error in parameter
Public Const canERR_NOMSG = -2          ' No messages available
Public Const canERR_NOTFOUND = -3       ' Specified hw not found
Public Const canERR_NOMEM = -4          ' Out of memory
Public Const canERR_NOCHANNELS = -5     ' No channels avaliable
Public Const canERR_RESERVED_3 = -6
Public Const canERR_TIMEOUT = -7        ' Timeout occurred
Public Const canERR_NOTINITIALIZED = -8 ' Lib not initialized
Public Const canERR_NOHANDLES = -9      ' Can't get handle
Public Const canERR_INVHANDLE = -10     ' Handle is invalid
Public Const canERR_INIFILE = -11       ' Error in the ini-file (16-bit only)
Public Const canERR_DRIVER = -12        ' CAN driver type not supported
Public Const canERR_TXBUFOFL = -13      ' Transmit buffer overflow
Public Const canERR_RESERVED_1 = -14
Public Const canERR_HARDWARE = -15      ' Some hardware error has occurred
Public Const canERR_DYNALOAD = -16      ' Can't find requested DLL
Public Const canERR_DYNALIB = -17       ' DLL seems to be wrong version
Public Const canERR_DYNAINIT = -18      ' Error when initializing DLL
Public Const canERR_RESERVED_4 = -19
Public Const canERR_RESERVED_5 = -20
Public Const canERR_RESERVED_6 = -21
Public Const canERR_RESERVED_2 = -22
Public Const canERR_DRIVERLOAD = -23    ' Can't find/load driver
Public Const canERR_DRIVERFAILED = -24  ' DeviceIOControl failed; use Win32
GetLastError()
Public Const canERR_NOCONFIGMGR = -25    ' Can't find req'd config s/w (e.g.
CS/SS)
Public Const canERR_NOCARD = -26        ' The card was removed or not inserted
Public Const canERR_RESERVED_7 = -27
```

```
Public Const canERR_REGISTRY = -28      ' Error in the Registry
Public Const canERR_LICENSE = -29       ' The license is not valid.
Public Const canERR_INTERNAL = -30      ' Internal error in the driver.
Public Const canERR_NO_ACCESS = -31           ' Access denied
Public Const canERR_NOT_IMPLEMENTED = -32


' Notification codes
Public Const canEVENT_RX = 32000              ' Receive event
Public Const canEVENT_TX = 32001              ' Transmit event
Public Const canEVENT_ERROR = 32002           ' Error event
Public Const canEVENT_STATUS = 32003                ' Change-of-status event

' Used in canSetNotify
Public Const canNOTIFY_NONE = 0                                 ' Turn notifications
off
Public Const canNOTIFY_RX = &H1                 ' Notify on receive
Public Const canNOTIFY_TX = &H2                 ' Notify on transmit
Public Const canNOTIFY_ERROR = &H4                ' Notify on error
Public Const canNOTIFY_STATUS = &H8                                     ' Notify on
(some) status changes

Public Const canSTAT_ERROR_PASSIVE = &H1                ' The circuit is error
passive
Public Const canSTAT_BUS_OFF = &H2              ' The circuit is Off Bus
Public Const canSTAT_ERROR_WARNING = &H4                ' At least one error
counter > 96
Public Const canSTAT_ERROR_ACTIVE = &H8                 ' The circuit is error
active.
Public Const canSTAT_TX_PENDING = &H10                  ' There are messages
pending transmission
Public Const canSTAT_RX_PENDING = &H20          ' There are messages in the
receive buffer
Public Const canSTAT_RESERVED_1 = &H40
Public Const canSTAT_TXERR = &H80               ' There has been at least one TX
error
Public Const canSTAT_RXERR = &H100                ' There has been at least one
RX error of some sort
Public Const canSTAT_HW_OVERRUN = &H200           ' The has been at least
one HW buffer overflow
Public Const canSTAT_SW_OVERRUN = &H400           ' The has been at least
one SW buffer overflow
Public Const canSTAT_OVERRUN = &H600

' Message information flags
Public Const canMSG_MASK = &HFF             ' Used to mask the non-info bits
Public Const canMSG_RTR = &H1               ' Message is a remote request
Public Const canMSG_STD = &H2               ' Message has a standard ID
```

```
Public Const canMSG_EXT = &H4                   ' Message has a extended ID
Public Const canMSG_WAKEUP = &H8                    ' Message was received in
wakeup mode
Public Const canMSG_NERR = &H10                     ' NERR was active during the
message
Public Const canMSG_ERROR_FRAME = &H20          ' Message is an error frame
Public Const canMSG_TXACK = &H40                ' Message is a TX REQUEST
Public Const canMSG_TXRQ = &H80                 ' Message is a TX ACK

' Message error flags, >= &H0100
Public Const canMSGERR_MASK = &HFF00            ' Used to mask the non-error
bits
Public Const canMSGERR_RXERR = &H100            ' Any RX error
Public Const canMSGERR_HW_OVERRUN = &H200       ' HW buffer overrun
Public Const canMSGERR_SW_OVERRUN = &H400       ' SW buffer overrun
Public Const canMSGERR_STUFF = &H800            ' Stuff error
Public Const canMSGERR_FORM = &H1000            ' Form error
Public Const canMSGERR_CRC = &H2000             ' CRC error
Public Const canMSGERR_BIT0 = &H4000            ' Sent dom, read rec
Public Const canMSGERR_BIT1 = &H8000            ' Sent rec, read dom

'
' Convenience values for the message error flags.
'
Public Const canMSGERR_OVERRUN = &H600          ' Any overrun condition.
Public Const canMSGERR_BIT = &HC000             ' Any bit error.
Public Const canMSGERR_BUSERR = &HF800          ' Any RX error


'
' canlib.h
'

Public Const canINVALID_HANDLE = -1

Public Const WM__CANLIB = (&H400 + 16354)     ' &H400 = WM_USER

' Obsolete
Public Const canCIRCUIT_ANY = -1                ' Any circuit will do
Public Const canCARD_ANY = -1                   ' Any card will do
Public Const canCHANNEL_ANY = -1                ' Any channel will do

' Flags for canOpenChannel
Public Const canWANT_EXCLUSIVE = &H8
Public Const canWANT_EXTENDED = &H10
Public Const canWANT_VIRTUAL = &H20
Public Const canOPEN_EXCLUSIVE = canWANT_EXCLUSIVE
Public Const canOPEN_REQUIRE_EXTENDED = canWANT_EXTENDED
```

```
Public Const canOPEN_ACCEPT_VIRTUAL = canWANT_VIRTUAL
Public Const canOPEN_OVERRIDE_EXCLUSIVE = &H40
Public Const canOPEN_REQUIRE_INIT_ACCESS = &H80
Public Const canOPEN_NO_INIT_ACCESS = &H100
Public Const canOPEN_ACCEPT_LARGE_DLC = &H200  ' DLC can be greater
than 8

' Flags for canAccept
Public Const canFILTER_ACCEPT = 1
Public Const canFILTER_REJECT = 2
Public Const canFILTER_SET_CODE_STD = 3
Public Const canFILTER_SET_MASK_STD = 4
Public Const canFILTER_SET_CODE_EXT = 5
Public Const canFILTER_SET_MASK_EXT = 6

Public Const canFILTER_NULL_MASK = 0

'
' CAN driver types - not all are supported on all cards.
'
Public Const canDRIVER_NORMAL = 4
Public Const canDRIVER_SILENT = 1
Public Const canDRIVER_SELFRECEPTION = 8
Public Const canDRIVER_OFF = 0

'
' Common bus speeds. Used in canSetBusParams.
' The values are translated in canlib, canTranslateBaud().
'
Public Const BAUD_1M = -1
Public Const BAUD_500K = -2
Public Const BAUD_250K = -3
Public Const BAUD_125K = -4
Public Const BAUD_100K = -5
Public Const BAUD_62K = -6
Public Const BAUD_50K = -7
Public Const BAUD_83K = -8

'
' IOCTL types
'
Public Const canIOCTL_PREFER_EXT = 1
Public Const canIOCTL_PREFER_STD = 2
' 3, 4 reserved.
Public Const canIOCTL_CLEAR_ERROR_COUNTERS = 5
Public Const canIOCTL_SET_TIMER_SCALE = 6
Public Const canIOCTL_SET_TXACK = 7
```

```
Type canHWDescr
  circuitType As Long          ' Any one of canCARD_XXX
  cardtype    As Long          ' Any one of PCCAN_xxx etc.
  channel     As Long          ' Channel # or canCHANNEL_ANY
End Type

Type canSWDescr
  rxBufSize   As Long
  txBufSize   As Long
  alloc       As Long          ' Should always be 0 when called from VB
  deAlloc     As Long          ' Should always be 0 when called from VB
End Type


Declare Sub canInitializeLibrary Lib "CANLIB32" ()

Declare Function canClose Lib "CANLIB32" (ByVal Handle As Long) As Long

Declare Function canBusOn Lib "CANLIB32" (ByVal Handle As Long) As Long

Declare Function canBusOff Lib "CANLIB32" (ByVal Handle As Long) As Long

Declare Function canSetBusParams Lib "CANLIB32" (ByVal Handle As Long,
ByVal freq As Long, ByVal tseg1 As Long, ByVal tseg2 As Long, ByVal sjw As
Long, ByVal noSamp As Long, ByVal syncMode As Long) As Long

Declare Function canGetBusParams Lib "CANLIB32" (ByVal Handle As Long,
ByRef freq As Long, ByRef tseg1 As Long, ByRef tseg2 As Long, ByRef sjw As
Long, ByRef noSamp As Long, ByRef syncMode As Long) As Long

Declare Function canSetBusOutputControl Lib "CANLIB32" (ByVal Handle As
Long, ByVal drivertype As Long) As Long

Declare Function canGetBusOutputControl Lib "CANLIB32" (ByVal Handle As
Long, ByRef drivertype As Long) As Long

Declare Function canAccept Lib "CANLIB32" (ByVal Handle As Long, ByVal
envelope As Long, ByVal flag As Long) As Long

Declare Function canReadStatus Lib "CANLIB32" (ByVal Handle As Long, ByRef
Flags As Long) As Long

Declare Function canReadErrorCounters Lib "CANLIB32" (ByVal Handle As Long,
ByRef txErr As Long, ByRef rxErr As Long, ByRef ovErr As Long) As Long

Declare Function canWrite Lib "CANLIB32" (ByVal Handle As Long, ByVal id As
Long, ByRef msg As Any, ByVal dlc As Long, ByVal flag As Long) As Long
```

Declare Function canWriteSync Lib "CANLIB32" (ByVal Handle As Long, ByVal timeout As Long) As Long

Declare Function canRead Lib "CANLIB32" (ByVal Handle As Long, ByRef id As Long, ByRef msg As Any, ByRef dlc As Long, ByRef flag As Long, ByRef time As Long) As Long

Declare Function canReadWait Lib "CANLIB32" (ByVal Handle As Long, ByRef id As Long, ByRef msg As Any, ByRef dlc As Long, ByRef flag As Long, ByRef time As Long, ByRef timeout As Long) As Long

Declare Function canReadSpecific Lib "CANLIB32" (ByVal Handle As Long, ByVal id As Long, ByRef msg As Any, ByRef dlc As Long, ByRef flag As Long, ByRef time As Long) As Long

Declare Function canReadSync Lib "CANLIB32" (ByVal Handle As Long, ByVal timeout As Long) As Long

Declare Function canReadSyncSpecific Lib "CANLIB32" (ByVal Handle As Long, ByVal id As Long, ByVal timeout As Long) As Long

Declare Function canReadSpecificSkip Lib "CANLIB32" (ByVal Handle As Long, ByVal id As Long, ByRef msg As Any, ByRef dlc As Long, ByRef flag As Long, ByRef time As Long) As Long

Declare Function canSetNotify Lib "CANLIB32" (ByVal Handle As Long, ByVal aHWnd As Long, ByVal aNotifyFlags As Long) As Long

Declare Function canTranslateBaud Lib "CANLIB32" (ByRef freq As Long, ByRef tseg1 As Long, ByRef tseg2 As Long, ByRef sjw As Long, ByRef noSamp As Long, ByRef syncMode As Long) As Long

Declare Function canGetErrorText Lib "CANLIB32" (ByVal Err As Long, ByRef Buf As Any, ByVal bufsiz As Long) As Long

Declare Function canGetVersion Lib "CANLIB32" () As Long

Declare Function canGetCircuits Lib "CANLIB32" (ByRef context As Long, ByRef name As String, ByRef vendor As String, ByRef version As String, ByRef cardtype As Long, ByRef circtype As Long, ByRef channel As Long) As Long

Declare Function canIoCtl Lib "CANLIB32" (ByVal Handle As Long, ByVal Func As Long, ByRef Buf As Any, ByVal Buflen As Long) As Long

Declare Function canReadTimer Lib "CANLIB32" (ByVal Handle As Long) As Long

```
Declare Function canOpenChannel Lib "CANLIB32" (ByVal Handle As Long, ByVal Flags As Long) As Long

' canlib32 specific functions
Declare Function canGetNumberOfChannels Lib "CANLIB32" (ByRef channelCount As Long) As Long

Declare Function canGetChannelData Lib "CANLIB32" (ByVal channel As Long, ByVal item As Long, ByRef buffer As Any, ByVal bufsize As Long) As Long

Public Const canCHANNELDATA_CHANNEL_CAP = 1
Public Const canCHANNELDATA_TRANS_CAP = 2
Public Const canCHANNELDATA_CHANNEL_FLAGS = 3          ' available, etc
Public Const canCHANNELDATA_CARD_TYPE = 4                              ' canHWTYPE_xxx
Public Const canCHANNELDATA_CARD_NUMBER = 5              ' Number in machine, 0,1,...
Public Const canCHANNELDATA_CHAN_NO_ON_CARD = 6
Public Const canCHANNELDATA_CARD_SERIAL_NO = 7
Public Const canCHANNELDATA_TRANS_SERIAL_NO = 8
Public Const canCHANNELDATA_CARD_FIRMWARE_REV = 9
Public Const canCHANNELDATA_CARD_HARDWARE_REV = 10
Public Const canCHANNELDATA_CARD_UPC_NO = 11
Public Const canCHANNELDATA_TRANS_UPC_NO = 12
Public Const canCHANNELDATA_CHANNEL_NAME = 13
Public Const canCHANNELDATA_DLL_FILE_VERSION = 14
Public Const canCHANNELDATA_DLL_PRODUCT_VERSION = 15
Public Const canCHANNELDATA_DLL_FILETYPE = 16
Public Const canCHANNELDATA_TRANS_TYPE = 17
Public Const canCHANNELDATA_DEVICE_PHYSICAL_POSITION = 18
Public Const canCHANNELDATA_UI_NUMBER = 19
Public Const canCHANNELDATA_TIMESYNC_ENABLED = 20
Public Const canCHANNELDATA_DRIVER_FILE_VERSION = 21
Public Const canCHANNELDATA_DRIVER_PRODUCT_VERSION = 22
Public Const canCHANNELDATA_MFGNAME_UNICODE = 23
Public Const canCHANNELDATA_MFGNAME_ASCII = 24
Public Const canCHANNELDATA_DEVDESCR_UNICODE = 25
Public Const canCHANNELDATA_DEVDESCR_ASCII = 26

' channelFlags in canChannelData
Public Const canCHANNEL_IS_EXCLUSIVE = &H1
Public Const canCHANNEL_IS_OPEN = &H2

' Hardware types.
Public Const canHWTYPE_NONE = 0            ' Unknown
Public Const canHWTYPE_VIRTUAL = 1          ' Virtual channel.
Public Const canHWTYPE_LAPCAN = 2           ' LAPcan
Public Const canHWTYPE_CANPARI = 3          ' CANpari
```

```
Public Const canHWTYPE_PCCAN = 8            ' PCcan-X
Public Const canHWTYPE_PCICAN = 9           ' PCIcan.
Public Const canHWTYPE_USBCAN = 11              ' USBcan Family and relatives
Public Const canHWTYPE_PCICAN_II = 40       ' PCIcan II family
Public Const canHWTYPE_USBCAN_II = 42       ' USBcan II, Memorator et al
Public Const canHWTYPE_SIMULATED = 44               ' SImulated CAN bus for
Creator
Public Const canHWTYPE_ACQUISITOR = 46      ' Acquisitor et al
Public Const canHWTYPE_LEAF = 48                ' Kvaser Leaf Family

' Channel capabilities.
Public Const canCHANNEL_CAP_EXTENDED_CAN = &H1
Public Const canCHANNEL_CAP_BUS_STATISTICS = &H2
Public Const canCHANNEL_CAP_ERROR_COUNTERS = &H4
Public Const canCHANNEL_CAP_CAN_DIAGNOSTICS = &H8
Public Const canCHANNEL_CAP_GENERATE_ERROR = &H10
Public Const canCHANNEL_CAP_GENERATE_OVERLOAD = &H20
Public Const canCHANNEL_CAP_TXREQUEST = &H40
Public Const canCHANNEL_CAP_TXACKNOWLEDGE = &H80
Public Const canCHANNEL_CAP_VIRTUAL = &H10000
Public Const canCHANNEL_CAP_SIMULATED = &H20000

' Driver (transceiver) capabilities
Public Const canDRIVER_CAP_HIGHSPEED = &H1

Public Const canIOCTL_GET_RX_BUFFER_LEVEL = 8
Public Const canIOCTL_GET_TX_BUFFER_LEVEL = 9
Public Const canIOCTL_FLUSH_RX_BUFFER = 10
Public Const canIOCTL_FLUSH_TX_BUFFER = 11
Public Const canIOCTL_GET_TIMER_SCALE = 12
Public Const canIOCTL_SET_TXRQ = 13
Public Const canIOCTL_GET_EVENTHANDLE = 14
Public Const canIOCTL_SET_BYPASS_MODE = 15
Public Const canIOCTL_SET_WAKEUP = 16
Public Const canIOCTL_GET_DRIVERHANDLE = 17
Public Const canIOCTL_MAP_RXQUEUE = 18
Public Const canIOCTL_GET_WAKEUP = 19
Public Const canIOCTL_SET_REPORT_ACCESS_ERRORS = 20
Public Const canIOCTL_GET_REPORT_ACCESS_ERRORS = 21
Public Const canIOCTL_CONNECT_TO_VIRTUAL_BUS = 22
Public Const canIOCTL_DISCONNECT_FROM_VIRTUAL_BUS = 23
Public Const canIOCTL_SET_USER_IOPORT = 24
Public Const canIOCTL_GET_USER_IOPORT = 25
Public Const canIOCTL_SET_BUFFER_WRAPAROUND_MODE = 26
Public Const canIOCTL_SET_RX_QUEUE_SIZE = 27

Type canUserIoPortData
  portNo As Long
```

```
   portValue As Long
End Type

Declare Function canWaitForEvent Lib "CANLIB32" (ByVal hnd As Long, ByVal
timeout As Long) As Long

Declare Function canSetBusParamsC200 Lib "CANLIB32" (ByVal hnd As Long,
ByVal btr0 As Long, ByVal btr1 As Long) As Long

Declare Function canSetDriverMode Lib "CANLIB32" (ByVal hnd As Long, ByVal
lineMode As Long, ByVal resNet As Long) As Long

Declare Function canGetDriverMode Lib "CANLIB32" (ByVal hnd As Long, ByRef
lineMode As Long, ByRef resNet As Long) As Long

Declare Function canGetVersionEx Lib "CANLIB32" (ByVal itemCode As Long)
As Long

Declare Function canParamGetCount Lib "CANLIB32" () As Long

Declare Function canParamCommitChanges Lib "CANLIB32" () As Long

Declare Function canParamDeleteEntry Lib "CANLIB32" (ByVal index As Long)
As Long

Declare Function canParamCreateNewEntry Lib "CANLIB32" () As Long

Declare Function canParamSwapEntries Lib "CANLIB32" (ByVal index1 As Long,
ByVal index2 As Long) As Long


' Must be treated like canGetErrorText -  i.e. define a local VB function
' that converts the string parameter. Left as an exercise to the reader.
' Declare Function canParamGetName Lib "CANLIB32" (ByVal index As Long,
ByVal buffer As String, ByVal maxlen As Long) As Long
' Declare Function canParamSetName Lib "CANLIB32" (ByVal index As Long,
ByVal buffer As String) As Long
' Declare Function canParamFindByName Lib "CANLIB32" (ByVal name As
String) As Long


Declare Function canParamGetChannelNumber Lib "CANLIB32" (ByVal index As
Long) As Long

Declare Function canParamGetBusParams Lib "CANLIB32" (ByVal index As Long,
ByRef bitrate As Long, ByRef tseg1 As Long, ByRef tseg2 As Long, ByRef sjw As
Long, ByRef noSamp As Long) As Long
```

```
Declare Function canParamSetChannelNumber Lib "CANLIB32" (ByVal index As Long, ByVal channel As Long) As Long

Declare Function canParamSetBusParams Lib "CANLIB32" (ByVal index As Long, ByVal bitrate As Long, ByVal tseg1 As Long, ByVal tseg2 As Long, ByVal sjw As Long, ByVal noSamp As Long) As Long

' Frees all object buffers associated with the specified handle.
Declare Function canObjBufFreeAll Lib "CANLIB32" (ByVal Handle As Long) As Long

' Allocates an object buffer of the specified type.
Declare Function canObjBufAllocate Lib "CANLIB32" (ByVal Handle As Long, ByVal tp As Long) As Long
Public Const canOBJBUF_TYPE_AUTO_RESPONSE = &H1
Public Const canOBJBUF_TYPE_PERIODIC_TX = &H2

' Deallocates the object buffer with the specified index.
Declare Function canObjBufFree Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long) As Long

' Writes CAN data to the object buffer with the specified index.
Declare Function canObjBufWrite Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long, ByVal id As Long, ByRef msg As Any, ByVal dlc As Long, ByVal Flags As Long) As Long

' For an AUTO_RESPONSE buffer, set the code and mask that together define
' the identifier(s) that trigger(s) the automatic response.
Declare Function canObjBufSetFilter Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long, ByVal code As Long, ByVal mask As Long) As Long

' Sets buffer-speficic flags.
Declare Function canObjBufSetFlags Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long, ByVal Flags As Long) As Long
' The buffer responds to RTRs only, not regular messages.
' AUTO_RESPONSE buffers only
Public Const canOBJBUF_AUTO_RESPONSE_RTR_ONLY = &H1

' Sets transmission period for auto tx buffers.
Declare Function canObjBufSetPeriod Lib "CANLIB32" (ByVal hnd As Long, ByVal idx As Long, ByVal period As Long) As Long

' Enable object buffer with index idx.
Declare Function canObjBufEnable Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long) As Long

' Disable object buffer with index idx.
```

```
Declare Function canObjBufDisable Lib "CANLIB32" (ByVal Handle As Long, ByVal idx As Long) As Long

' For certain diagnostics.
Declare Function canObjBufSendBurst Lib "CANLIB32" (ByVal hnd As Long, ByVal idx As Long, ByVal burstlen As Long) As Long

' Check for specific version(s) of CANLIB.
Public Const canVERSION_DONT_ACCEPT_LATER = &H1
Public Const canVERSION_DONT_ACCEPT_BETAS = &H2
Declare Function canProbeVersion Lib "CANLIB32" (ByVal hnd As Long, ByVal major As Long, ByVal minor As Long, ByVal oem_id As Long, ByVal Flags As Long) As Boolean

' Try to "reset" the CAN bus.
Declare Function canResetBus Lib "CANLIB32" (ByVal Handle As Long) As Long

' Convenience function that combines canWrite and canWriteSync.
Declare Function canWriteWait Lib "CANLIB32" (ByVal Handle As Long, ByVal id As Long, ByRef msg As Any, ByVal dlc As Long, ByVal flag As Long, ByVal timeout As Long) As Long

' Tell canlib32.dll to unload its DLLs.
Declare Function canUnloadLibrary Lib "CANLIB32" () As Long

Declare Function canSetAcceptanceFilter Lib "CANLIB32" (ByVal hnd As Long, ByVal code As Long, ByVal mask As Long, ByVal is_extended As Long) As Long



'------------------------------------------------------------------------
' Obsolete definitions follow.
'------------------------------------------------------------------------

' Old-style Circuit status flags
Public Const canCIRCSTAT_ERROR_PASSIVE = &H1       ' Error passive
Public Const canCIRCSTAT_BUS_OFF = &H2             ' Bus off
Public Const canCIRCSTAT_ERROR_WARNING = &H4       ' Error counter > 96


' Circuit types.
Public Const PCCAN_PHILIPS = 1                 ' 82C200 on PCCAN 1.0
Public Const PCCAN_INTEL526 = 2                ' Not supported.
Public Const PCCAN_INTEL527 = 3                ' 82527 on PCCAN 1.0
Public Const CANCARD_NEC72005 = 4              ' NEC72005 on CANCard

' Card types.
Public Const canCARD_PCCAN = 1                 ' PCCAN ver 1.x (KVASER)
```

```
Public Const canCARD_CANCARD = 2                ' CANCard (Softing)
Public Const canCARD_AC2 = 3                    ' CAN-AC2 (Softing)


Public Const canWANT_ACTIONS = &H1
Public Const canWANT_OWN_BUFFERS = &H2
Public Const canWANT_ERROR_COUNTERS = &H4

' The canFlgXXX are left for compatibility.
Public Const canFlgACCEPT = 1
Public Const canFlgREJECT = 2
Public Const canFlgCODE = 3
Public Const canFlgMASK = 4

' Flags for action routines
Public Const canDISCARD_MESSAGE = 3
Public Const canRETAIN_MESSAGE = 4

' For busParams - sync on rising edge only or both rising and falling edge
Public Const canSLOW_MODE = &H1 ' Sync on rising and falling edge

'
' CAN driver types; these constants are retained for compatibility.
'
Public Const canOFF = 0
Public Const canTRISTATE = 1
Public Const canPULLUP = 2
Public Const canPULLDOWN = 3
Public Const canPUSHPULL = 4                    ' This is the usual setting.
Public Const canINVPULLUP = 5
Public Const canINVPULLDOWN = 6
Public Const canINVPUSHPULL = 7

Declare Function canInstallAction Lib "CANLIB32" (ByVal Handle As Long,
ByVal id As Long, ByRef Func As Long) As Long

Declare Function canUninstallAction Lib "CANLIB32" (ByVal Handle As Long,
ByRef id As Long) As Long

Declare Function canInstallOwnBuffer Lib "CANLIB32" (ByVal Handle As Long,
ByRef id As Long, ByRef length As Long, ByRef Buf As String) As Long

Declare Function canUninstallOwnBuffer Lib "CANLIB32" (ByVal Handle As
Long, ByRef id As Long) As Long

Public Const canIOCTL_LOCAL_ECHO_ON = 3
Public Const canIOCTL_LOCAL_ECHO_OFF = 4
```

```
' Not obsolete.. but new program should use canInitializeLibrary
Declare Function canLocateHardware Lib "CANLIB32" () As Long

' Not obsolete.. but new programs should use canOpenChannel instead
Declare Function canOpen Lib "CANLIB32" (ByRef hwdescr As canHWDescr,
ByRef swdescr As canSWDescr, ByVal Flags As Long) As Long

' ------------------------------------------------------------------------------
' VB Special fixes below
' ------------------------------------------------------------------------------
Declare Function GetErrorText Lib "CANLIB32" Alias "canGetErrorText" (ByVal
Err As Long, ByVal Buf As String, ByVal bufsiz As Long) As Long

'
' Use this one instead of canGetErrorText to get the error text for
' a status code
'
Public Function GetErrorTextVB(ByVal Err As Long) As String
  Dim Buf As String * 200
  Dim stat As Integer
  stat = GetErrorText(Err, Buf, 200)
  Buf = Left(Buf, InStr(1, Buf, Chr(0), vbBinaryCompare) - 1)
  GetErrorTextVB = Buf
End Function
```

Form

```
Dim oku As Byte
Dim Handle As Long
Dim adet As Integer

Private Sub DisplayStatus(stat As Long)
   ' Label2.Caption = Str(stat)
   Label2.Caption = GetErrorTextVB(stat)
End Sub

Private Sub Command1_Click()
canInitializeLibrary
End Sub

Private Sub command2_Click()
ayar = InputBox("Kullanılacak CAN Portunu Seçiniz", "CAN Bus Programlayıcı")
Handle = canOpenChannel(Val(ayar), canWANT_EXCLUSIVE)
   If (Handle < 0) Then
      DisplayStatus (Handle)
   Else
      Label2.Caption = "Tamam"
```

```
    End If
End Sub

Private Sub Command3_Click()
Dim stat As Long
    stat = canSetBusParams(Handle, BAUD_250K, 0, 0, 0, 0, 0)
    DisplayStatus (stat)
    stat = canSetBusOutputControl(Handle, canPUSHPULL)
    DisplayStatus (stat)
End Sub

Private Sub Command4_Click()
 Dim stat As Long
    stat = canBusOn(Handle)
    DisplayStatus (stat)
End Sub

Private Sub Command5_Click()
Dim stat As Long
  Dim msg As Byte
  Open App.Path & "\son.txt" For Input As 1
  Do While Not EOF(1)
  Line Input #1, ss
  adet = adet + 1
  Loop
  Close #1
  Open App.Path & "\son.txt" For Input As 1
  Timer1.Enabled = True

End Sub

Private Sub Command6_Click()
Dim stat As Long
    Dim id As Long
    Dim msg(0 To 7) As Byte
    Dim dlc As Long
    Dim Flags As Long
    Dim tmp As String
    Dim time As Long
    Dim i As Integer
    stat = canRead(Handle, id, msg(0), dlc, Flags, time)
    If stat = canOK Then
        tmp = ""
        tmp = tmp + "Id=" + Str(id)
        tmp = tmp + " Data="
        For i = 0 To dlc - 1
            tmp = tmp + " " + Str(msg(i))
        Next i
```

```
        tmp = tmp + " DLC=" + Str(dlc)
        tmp = tmp + " Flags=" + Str(Flags)
        Label1.Caption = tmp
    ElseIf stat = canERR_NOMSG Then
      Label2.Caption = "Okunacak Mesaj Yok"
    Else
      ' an error
      Label2.Caption = "Hata Oluştu"
    End If
    DisplayStatus (stat)
End Sub

Private Sub Command7_Click()
Dim stat As Long
    stat = canBusOff(Handle)
    DisplayStatus (stat)

End Sub

Private Sub Command8_Click()
canClose (Handle)
    Handle = canINVALID_HANDLE
    End
End Sub

Private Sub Command9_Click()
CommonDialog1.ShowOpen
dosya = CommonDialog1.FileName

Open dosya For Input As 1
Open App.Path & "\ara.txt" For Output As 2
Do While Not EOF(1)
Line Input #1, satir
List1.AddItem satir
y = Mid(satir, 8, 2)
If y = "00" Then
say = say + 1
kac = Len(satir)
al = Mid(satir, 10, kac - 11)
Print #2, al
End If
Loop
Close #1, 2
Open App.Path & "\ara.txt" For Input As 1
Open App.Path & "\son.txt" For Output As 2
Do While Not EOF(1)
kontrol = kontrol + 1
Line Input #1, satir
```

```
kac = Len(satir)
If kontrol < say Then
For i = 1 To kac Step 2
ver = Mid(satir, i, 2)
oku = "&h" & ver
Print #2, oku
Next i
End If
Loop
Close #1, 2
Open App.Path & "\son.txt" For Input As 1
Do While Not EOF(1)
Line Input #1, satir
s = s + 1
f = f + satir
If s = 20 Then
s = 0
f = f + Chr(13) + Chr(10)
End If
Loop
RichTextBox1.Text = f
Close #1
End Sub


Private Sub Timer1_Timer()
Static a As Integer
a = a + 1
Line Input #1, al
oku = Val(al)
stat = canWrite(Handle, 1, oku, 1, 0)
Label2.Caption = Label2.Caption + al
If a = adet Then
Timer1.Enabled = False
Close #1
stat = canWrite(Handle, 2, 0, 1, 0)
End If
End Sub
```

EK – 2

Bellek Programlayıcı Programı Kodları

```
program CANUser_Bootloader

dim xx as boolean
dim aa  as byte
dim aa1 as byte
dim lenn as byte
dim aa2 as byte
dim gelen as byte[8]
dim topla as byte[63]
dim id as longint
dim zr as byte
dim i,adres as integer

 main:
  i=0
  TRISD = 0
  PORTD = 0
  aa = 0
  aa1 = 0
  aa2 = 0
  adres=$0D00
  xx=true
  aa1 =   CAN_TX_PRIORITY_0 and
          CAN_TX_XTD_FRAME and
          CAN_TX_NO_RTR_FRAME

  aa2 =   CAN_RX_FILTER_1 AND
          CAN_RX_XTD_FRAME


  aa =    CAN_CONFIG_SAMPLE_THRICE and
          CAN_CONFIG_PHSEG2_PRG_ON and
          CAN_CONFIG_STD_MSG and

          CAN_CONFIG_DBL_BUFFER_ON and
          CAN_CONFIG_VALID_STD_MSG and
          CAN_CONFIG_LINE_FILTER_OFF
  gelen[0]=0
  topla[0]=0
  CANInitialize( 1,1,3,3,1,aa)
  CANSetOperationMode(CAN_MODE_CONFIG,TRUE)
  CANSetBaudRate(1, 1, 3, 3, 1, aa)
  ID=-1
  CANSetMask(CAN_MASK_B1,ID,CAN_CONFIG_STD_MSG)
  CANSetMask(CAN_MASK_B2,ID,CAN_CONFIG_STD_MSG)
  CANSetFilter(CAN_FILTER_B1_F1,1,CAN_CONFIG_STD_MSG)
  CANSetFilter(CAN_FILTER_B1_F2,2,CAN_CONFIG_STD_MSG)
  CANSetOperationMode(CAN_MODE_NORMAL,TRUE)
```

```
while xx=true
   delay_ms(1000)
   zr = CANRead(id, gelen,lenn, aa2)
   if id=1 then
      topla[i]=gelen[0]
      i=i+1
      if i=64 then
        Flash_Write(adres,topla)
        adres=adres+64
        i=0
      end if
  end if
  if id=2 then
        xx=false
  end if
wend
   if i<64 then
 Flash_Write(adres,topla)

   end if
end.
```

EK – 3

Kullanıcı Uygulama Programı Kodları

```
program Led_blinking
main:
    TRISd = 0
    PORTd = $FF
    Delay_ms(5000)
    Portd =$00
    Delay_ms(1000)
    Portd=$FF
   Delay_ms(5000)
   PORTd =$00
end.
```

EK – 4

Kullanıcı Uygulama Programı
(Heksadesimal Form)

```
:1000000004EF00F0FFFFFFFF956AFF0E836E1A0EEC
:100010000C6EFF0E0B6EFF0E0A6E0C2E01D007D079
:100020000B2E01D003D00A2EFED7FAD7F6D77F0EBB
:100030000B6EFF0E0A6E0B2E01D003D00A2EFED7D8
:10004000FAD7580E0A6E0A2EFED700000000836A07
:10005000060E0C6EFF0E0B6EFF0E0A6E0C2E01D0FC
:1000600007D00B2E01D003D00A2EFED7FAD7F6D731
:100070001A0E0B6EFF0E0A6E0B2E01D003D00A2E45
:10008000FED7FAD7420E0A6E0A2EFED7FF0E836EF7
:100090001A0E0C6EFF0E0B6EFF0E0A6E0C2E01D0A8
:1000A00007D00B2E01D003D00A2EFED7FAD7F6D7F1
:1000B0007F0E0B6EFF0E0A6E0B2E01D003D00A2EA0
:1000C000FED7FAD7580E0A6E0A2EFED7000000009F
:1000D000836AFFD7FFFFFFFFFFFFFFFFFFFFFF69
:020000040030CA
:0E000000FFFAFDFEFFFFFAFFFFFFFFFFFFFF0D
:00000001FF
```

EK – 5

Kullanıcı Uygulama Programının CAN Mesajı Data Alanı İçerisindeki Değerleri

4 239 0 240 255 255 255 255 149 106 255 14 131 110 26 14 12 110 255 14 11 110 255 14 10 110 12 46 1 208 7 208 11 46 1 208 3 208 10 46 254 215 250 215 246 215 127 14 11 110 255 14 10 110 11 46 1 208 3 208 10 46 254 215 250 215 88 14 10 110 10 46 254 215 0 0 0 0 131 106 6 14 12 110 255 14 11 110 255 14 10 110 12 46 1 208 7 208 11 46 1 208 3 208 10 46 254 215 250 215 246 215 26 14 11 110 255 14 10 110 11 46 1 208 3 208 10 46 254 215 250 215 66 14 10 110 10 46 254 215 255 14 131 110 26 14 12 110 255 14 11 110 255 14 10 110 12 46 1 208 7 208 11 46 1 208 3 208 10 46 254 215 250 215 246 215 127 14 11 110 255 14 10 110 11 46 1 208 3 208 10 46 254 215 250 215 88 14 10 110 10 46 254 215 0 0 0 0 131 106 255 215 255 255 255 255 255 255 255 255 255 255 255 255

ÖZGEÇMİŞ

Adı Soyadı : İlker ÜNAL
Doğum Yeri : Ankara
Doğum Yılı : 1974
Medeni Hali : Evli

Eğitim Ve Akademik Durumu:

Lise	**:**	1988 – 1992	Isparta Teknik Lise – Bilgisayar Bölümü
Lisans	**:**	1992-1997	Marmara Üniversitesi Teknik Eğitim Fakültesi Elektronik ve Bilgisayar Öğretmenliği

Yabancı Dil : İnglizce

İş Deneyimi

1997-	Süleyman Demirel Üniversitesi Bucak Hikmet Tolunay Meslek Yüksekokulu – Öğretim Görevlisi