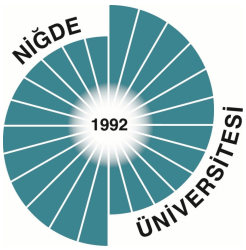
E. DİNÇER, 2010



T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

CAN BUS İLE DAĞITIK KONTROL UYGULAMASI

YÜKSEK LİSANS TEZİ

ERHAN DİNÇER

Şubat 2010

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

CAN BUS İLE DAĞITIK KONTROL UYGULAMASI

ERHAN DİNÇER

Yüksek Lisans Tezi

Danışman

Prof. Dr. Murat UZAM

Şubat 2010

ÖZET

CAN BUS İLE DAĞITIK KONTROL UYGULAMASI

DİNÇER, Erhan
Niğde Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman : Prof. Dr. Murat UZAM

Şubat 2010, 91 sayfa

Bu çalışmanın amacı CAN-Bus (Kontrol Alan Ağı Veri Yolu) teknolojisini kullanarak bir dağıtık kontrol uygulamasının tasarlanması ve gerçekleştirilmesidir.

CAN-Bus otomotiv endüstrisinde kullanılan en yaygın haberleşme sistemidir. Her ne kadar başlangıçta sadece otomotiv uygulamaları için tasarlanmış olsa da yüksek performansı ve üstün karakteristik özellikleri sebebiyle günümüzde birçok dağıtık endüstriyel kontrol uygulamasında yaygın olarak kullanılmaktadır.

Bu çalışmada dağıtık sistem olarak dört katlı model asansör kullanılmıştır. Sistemi dağıtık olarak kontrol etmek için, altı kontrol modülü tasarlanmıştır. Her bir modülde kontrol ve haberleşmeyi sağlamak için bir PIC16F877 mikrodenetleyicisi kullanılmıştır. Her bir modülün diğer modüllerle CAN-Bus üzerinden haberleşmesi için de bir MCP2515 CAN-SPI ve bir MCP2551 CAN-Bus çevirici ünitesinden oluşan bir CAN modülü gerçekleştirilmiştir. Böylece model asansörün CAN-Bus teknolojisi kullanılarak dağıtık kontrolü başarılı bir şekilde gerçekleştirilmiştir.

Anahtar sözcükler: CAN-Bus, Dağıtık Sistemler, İletişim Sistemleri, Otomasyon, Otomasyon Sistemleri, Kontrol Ağları, Mikrodenetleyiciler

SUMMARY

A DISTRIBUTED CONTROL APPLICATION USING CAN BUS

DİNÇER, Erhan

Nigde University

Graduate School of Natural and Applied Sciences

Department of Electric-Electronic Engineering

Supervisor : Professor Dr. Murat UZAM

February 2010, 91 pages

The aim of this study is to design and implement a distributed control application by using CAN-Bus (Controller Area Network Bus) technology.

CAN-Bus is the most widely used communication system in automotive industry. Although initially it is designed to be used for automotive applications, nowadays it is widely used in distributed industrial control applications due to its high performance and excellent characteristic specifications.

In this study, a model elevator with four layers is used as a distributed system. In order to control this system in a distributed manner six control modules are designed. In each control module a PIC16F877 microcontroller is used to carry out control and communication tasks. For establishing the communication of each control module with others by means of CAN-Bus a CAN module, consisting of an MCP2515 CAN-SPI and an MCP2551 CAN-Bus converter unit, is designed. Finally the distributed control of the model elevator system has been accomplished successfully.

Keywords: CAN-Bus, Distributed System, Communication Systems, Automation, Automation Systems, Control Networks, Microcontroller

TEŞEKKÜR

Bu yüksek lisans tezinin hazırlanmasında başta manevi desteklerini esirgemeyen sevgili eşim Özlem DİNÇER'e sonsuz teşekkür ederim.

Bu yüksek lisans tezinin oluşturulmasında emeği geçen sevgili hocam Prof. Dr. Murat UZAM'a, sevgili dostum Metin YURDANUR'a, bilgi ve becerileriyle desteğini esirgemeyen sevgili dostum Erdal URAS'a, malzeme tespiti konusunda yardımlarını esirgemeyen sevgili dostum Emrah İŞSEVER' e teşekkürlerimi, sevgi ve saygılarımı sunarım.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

FOTOĞRAFLAR DİZİNİ

KISALTMA VE SİMGELER

KISALTMA/SİMGE

CAN	Controller Area Network (Kontrol Alan Ağı)
CAN-Bus	Controller Area Network Bus(Kontrol Alan Ağı Veri Yolu)
PIC	Peripheral Interface Controller (Çevresel Denetleyici Arayüzü)
SCI	Serial Communications Interface (Seri İletişim Arayüzü)
USART	Universal Synchronous Asynchronous Receiver / Transmitter (Çok Amaçlı Senkron/Asenkron Alıcı / Verici)
A/S	Analog / Sayısal
S/A	Sayısal / Analog
MSSP	Master Synchronous Serial Port (Ana Senkronizasyonlu Seri Port)
PLC	Programmable Logic Controller (Programlanabilir Lojik Denetleyici)
OSI	Open System Interconnect (Açık Sistem Bağlantısı)
SPI	Serial Peripheral Interface (Seri Çevresel Arayüz)
CSMA	Carrier Sense Multiple Access (Çoklu Taşıyıcı Giriş Duyusu)
CD	Collision Detection (Çakışma Bulma)
CR	Collision Resolution (Çakışma Çözümü)
RTR	Remote Transmit Request (Uzak İleti Talebi)
CRC	Cyclic Redundancy Check (Periyodik Fazlalık Testi)
TEC	Transmit Error Counter (İletim Hata Sayıcısı)
REC	Receiver Error Counter (Alıcı Hata Sayıcısı)
DKS	Dağıtık Kontrol Sistemi
ACK	Acknowledge (Teyit)
I^2C	Inter-Integrated Circuit (Bütünleşik Devreler)

BÖLÜM I

GİRİŞ

Endüstriyel haberleşme sistemleri içerisinde geniş bir uygulama alanı bulan FieldBus (Alansal Veriyolu) ağları, özellikle bilgisayarla bütünleşik üretim zincirinin daha düşük seviyelerinde, sensörler, eyleyiciler (actuators) ve süreç denetleyicilerin ara bağlantısı için tasarlanmıştır.

FieldBus, bir süreç kontrol uygulamasının kontrol sistemleri ile saha elemanları ara bağlantısını sağlamak üzere kullanılan sayısal haberleşme hattıdır. FIP, Profibus, IEC/ISA ve CAN yaygın olarak tercih edilen FieldBus haberleşme sistemleridir. Bunlar arasındaki temel fark fiziksel ortama erişim mekanizmalarından kaynaklanmaktadır.

CAN, otomotiv endüstrisinde kullanılan en yaygın haberleşme sistemidir. Her ne kadar başlangıçta yalnızca otomotiv uygulamaları için tasarlanmış olsa da yüksek performansı ve üstün karakteristik özelliklerinden dolayı birçok dağıtık endüstriyel kontrol uygulamalarında yaygın olarak kullanılmaktadır. Öncelikli erişim mekanizması esasına göre çalışan CAN, gerçek zamanlı bir seri haberleşme protokolüne sahiptir. Bu yöntem, mesaj çarpışmamasını garanti etmekle beraber, iletişim hattının uzunluğunu sınırlandırır. Dolayısıyla, CAN düğümler 1 Mbit/s veri iletim hızı ile 40 m ve 100 Kbit/s veri iletim hızı ile de 500 m'lik bir veri yolu üzerinden bağlanabilirler.

Multi-Master (Çok Yöneticili) yani bütün CAN düğümlerinin veri iletebildiği ve bir kaçının da eş zamanlı olarak istekte bulunabildiği veri yolu sistemi olan CAN, hiçbir abone ya da kullanıcı için her hangi bir adreslemeye sahip olmamakla birlikte öncelikli mesajın iletilmesi şeklinde veri iletir. Bu veri iletişim protokolü, en sık Otomotiv ve Medikal endüstrisinde kullanım alanı bulmaktadır.

Günümüzde CAN'in kullanıldığı değişik ağ uygulamalarına akıllı motor kontrolü, robot kontrolü, akıllı sensörler, asansörler, akıllı binalar ve laboratuvar otomasyonu örnek olarak gösterilebilir.

CAN'in özellikleri; yüksek hızlı seri arabirim, düşük maliyetli fiziksel ortam, kısa yük büyüklüğü, hızlı tepki zamanları ve yüksek hata bulma ve düzeltme seviyesi olarak özetlenebilir.

CAN-Bus protokolü öncelikli uygulama alanı olan otomotiv endüstrisinde bilgi alış-verişi alanında bir standart olarak kabul edilmiş ve günümüzde endüstriyel otomasyon alanında da yaygın olarak kullanılmaya başlanmıştır. Bu başarı, geliştirilen yazılımın ve CAN ağında kullanılan haberleşme protokolünün sağlam yapıda olmasından kaynaklanmaktadır [1]. CAN-Bus protokolünü destekleyen pek çok ürün mevcuttur. Bu çalışmada kullanımı kolay olan ve ucuz ürünleriyle dikkat çeken Microchip [2] firmasının ürünlerinden CAN-Bus protokolünü destekleyen ürünler kullanılmıştır. Microchip firmasının üretmiş olduğu PIC mikrodenetleyicileri [2, 3]; flash tabanlı program belleği sayesinde defalarca programlanabilmeleri, ucuz olmaları, program geliştirmek ve yüklemek için gerekli olan donanım ve yazılımın kolay bir şekilde ve ucuz olarak elde edilebilmesi gibi nedenlerden dolayı kontrol uygulama mekanizması olarak seçilmiştir. Özellikle 18 serisi olarak adlandırılan PIC mikrodenetleyiciler, 40 MHz'e kadar çıkan saat frekansı ve 64 Kbyte'a kadar olan programlanabilir belleği ile bu alanda önemli bir yere sahiptir. CAN-Bus teknolojisini destekleyen ürünler de yine microchip firması tarafından üretilmektedir [4-6].

Bu Yüksek Lisans Tez çalışmasında, dört katlı model asansör üzerinde bir dağıtık kontrol uygulaması gerkçekleştirilmiştir. Asansör kontrol sistemleri çok teferruatlı, katı gerçek zamanlı sistemlerdir. Belirlenen zamanlama kıstasları içinde çeşitli işlemleri gerçekleştirmelidirler. Asansör denetleyici sistem, asansörün o andaki konumunu saptamalı ve kabin hızına bağlı olan süreç içerisinde verileri işlemelidir. Sistemde kontrol edilmesi gereken bileşenlerin çokluğu ve bazı uygulamalar için gerekli olan zaman kısıtlamaları bu tür sistemler üzerinde gerçek zamanlı işletim sistemlerinin kullanılmasını teşvik etmektedir. Temel olarak gerçek zamanlı kavramı bir program çevresinde oluşan olaylara belirlenen bir süre içerisinde cevap vermesi anlamına gelmektedir. Bu gibi sistemler olay güdümlü çalışan sistemler olarak belirtilmektedir ve gecikme kavramı ile ön plana çıkmaktadır. Bu sistemlerin çalışmasında iletişim çok önemlidir [29]. Çok iyi ağ kararlılığına sahip olması, cihazlar arası haberleşmede esneklik sağlaması ve maliyetinin düşüklüğü sebepleriyle günümüz asansör

sistemlerinde CAN-Bus teknolojisi büyük çoğunluklukla tercih edilmektedir. Ayrıca elektromanyetik uyumluluğun iyi olması ve çoğu mikrodenetleyicide bir CAN ara yüzü bulunması CAN-Bus teknolojisinin asansör sistemlerinde kullanılırlığını artırmıştır.

Bu Yüksek Lisans Tez çalışmasında, günümüzde otomasyon sistemlerinde yaygın olarak kullanılmaya başlanan CAN teknolojisi incelenmektedir. CAN teknolojisi ve mikrodenetleyici altyapısı yardımıyla bir dağıtık kontrol sisteminin tasarımı ve uygulaması gerçekleştirilmiştir.

Bu tez çalışmasının 2. bölümünde genelde PIC Mikrodenetleyicileri ve özelde PIC16F877 mikrodenetleyicisi incelenmektedir. 3. bölümde bu tez çalışmasında program geliştirmek için kullanılan MicroBasic Programlama Dili özet olarak incelenmiştir. CAN-Bus protokolü ile ilgili bir inceleme 4. bölümde sunulmuştur. Dağıtık kontrol uygulaması olarak seçilmiş olan dört katlı model asansör 5. bölümde anlatılmıştır. CAN-Bus kullanılarak gerçekleştirilen dört katlı model asansörün dağıtık kontrolu 6. bölümde açıklanmıştır. 7. ve son bölümde gerçekleştirilen bu tez çalışması sonunda elde edilen sonuçlar yer almaktadır.

BÖLÜM II

PIC MİKRODENETLEYİCİLERİ VE PIC 16F877

Bu Yüksek Lisans Tez çalışmasında dağıtık kontrolunun gerçekleştirilmesinde mikrodenetleyiciler kullanılmıştır. Günümüz elektronik uygulamalarındaki fonksiyonelliği şüphe götürmeyen PIC mikrodenetleyicileri içinde yaygın olarak tercih edilen ve bu bölümde detaylı olarak anlatılan PIC16F877 mikrodenetleyicisi tercih edilmiştir.

Tezin bu kısmında dağıtık kontrol uygulamasında PIC16F877 mikrodenetleyicisinin teknik özelliklerinden kısaca bahsedilecektir.

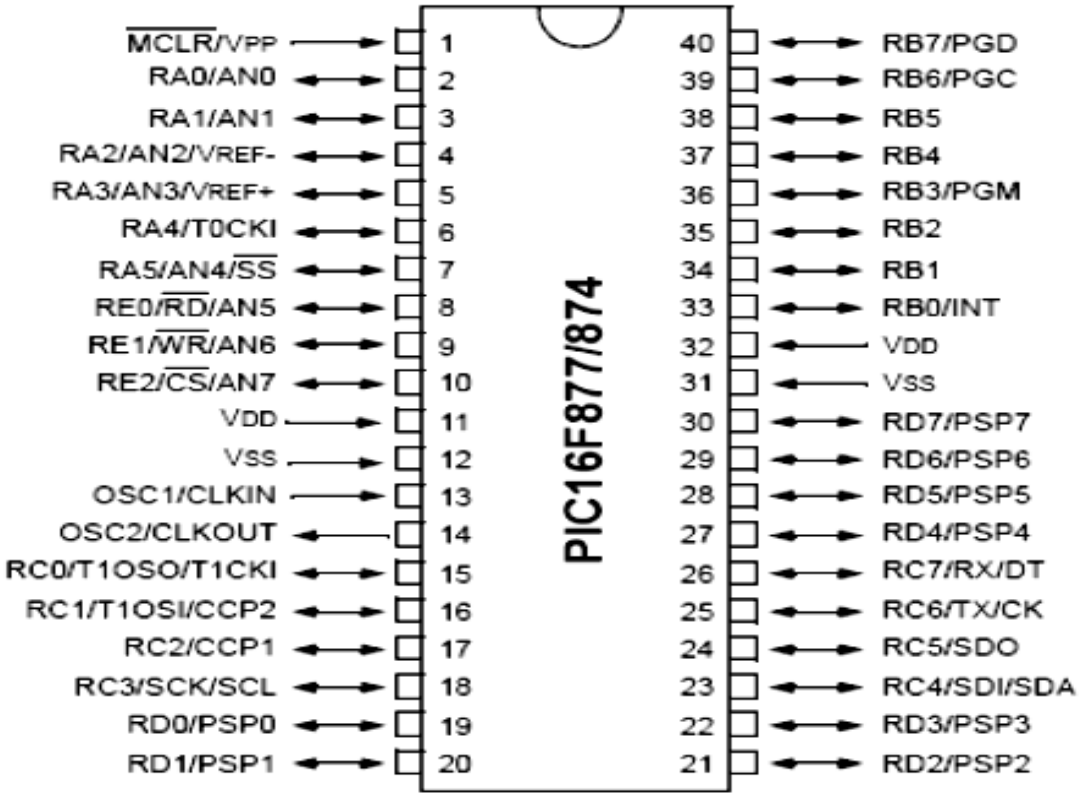
2.1 PIC16F877 Mikrodenetleyici Özellikleri ve Yapısı

PIC serisi mikroişlemciler Microchip (www.microchip.com) firması tarafından geliştirilmiş ve üretim amacı çok fonksiyonlu mantıksal uygulamaların hızlı ve ucuz bir mikrodenetleyici ile yazılım yoluyla karşılanmasıdır. PIC mikrodenetleyicilerinin mimarisi değiştirilmiş Harvard RISC yapısına dayalıdır [7, 8].

Gerçekleştirilen bu çalışmada PIC16F877 mikrodenetleyicisi, sistemlerin veri girişini sağlamak, sensör bilgilerini gönderip almak, motor kontrolünü sağlamak ve alınan bilgileri displayde göstermek ve CAN kontrol kartıyla haberleşip, yorumlamak amacıyla kullanılmıştır. Mikrodenetleyicinin bu işlemleri sağlaması için yeterli sayıda giriş/çıkış portu olmalıdır.

PIC16F877, PIC16F84 kadar popüler bir mikrodenetleyicidir. Flash ROM belleği 8Kword büyüklüğündedir. Bu flash belleği 1 milyon kez programlanabilir. 368 Byte veri belleği ve 256 Byte EEPROM (Erasable Programmable Read Only Memory) veri belleği vardır. Yüksek hızlı RISC (Reduced Instruction Set Computer) temeline dayalıdır. İşlem hızı DC-20 MHz'dir. 8 bit veri yoluna sahiptir. Aynı anda, veri belleğine 8 bit genişliğindeki bu yolla erişilirken, program belleğine program yolu ya da adres yolu denilen 14 bit genişliğindeki başka bir yolla erişilir. 35 adet komuta sahiptir

ve bu komutların her biri sadece 1 çevrimlik süre içerir [7, 8]. Şekil 2.1'de PIC16F877'nin bacak bağlantı yapısı görülmektedir. Çizelge 2.1'de ise PIC16F877' nin bacak açıklamaları verilmiştir.



Şekil 2.1 PIC16F877'nin bacak bağlantı yapısı

2.2 PIC 16F877'nin Port Yapısı ve Fonksiyonları

PIC 16F877'de A, B, C, D, E giriş/çıkış portları mevcuttur. Bu portların giriş ya da çıkış olarak kullanılması isteğe bağlıdır. Bu portlar giriş/çıkış işlemi gerçekleştirmekle beraber bazı çevresel işlemleri de yapacak özelliğe sahiptirler. Bu bacaklar çevre birimleri kullanıldığında genel giriş/çıkış olarak kullanılamaz.

2.2.1 Port A

Her bir biti bağımsız olarak giriş veya çıkış olarak ayarlanabilmektedir. A port'u hem sayısal giriş/çıkış hem de analog giriş/çıkış uçları görevi yapan 6'bitlik bir porttur.

Çizelge 2.1 PIC16F877'nin bacak açıklamaları

OSC1/CLKIN	13	G	ST/CMOS	Kristal osilatör girişi/Harici osilatör kaynağı girişi
OSC2/CLKOUT	14	Ç	—	Kristal osilatör çıkışı.RC osilatör modunda 1/4 f değerinde frekans çıkışı
MCLR/VPP/THV	1	G/P	ST	Mikrodenetleyici için reset ucu. Normal çalışmada 1 seviyesinde tutulur
RA0/AN0	2	G/Ç	TTL	PORTA:G/Ç olarak yönlendirilebilir port.Pinler G/Ç görevi dışında; RA0: 0. Analog giriş görevi yapar. RA1: 1. Analog giriş görevi yapar. RA2: 2. Analog giriş veya negatif referans gerilimi girişim görevi yapar. RA3: 3. Analog giriş veya pozitif referans gerilimi girişim görevi yapar. RA4: TIMER0 için clock girişi görevi yapar. Açık drain çıkışa sahiptir. RA5: 4. Analog giriş veya SSP için slave seçimi görevi yapar.
RA1/AN1	3	G/Ç	TTL	
RA2/AN2/VREF-	4	G/Ç	TTL	
RA3/AN3/VREF+	5	G/Ç	TTL	
RA4/T0CKI	6	G/Ç	ST	
RA5/SS/AN4	7	G/Ç	TTL	
RB0/INT	33	G/Ç	TTL/ST	PORTB: G/Ç olarak yönlendirilebilir port.Tüm girişleinde yazılımla programlanabilir düşük değerli pull-uplar vardır.Pinler G/Ç görevi dışında; RB0:Harici kesme ucu görevi yapar. RB3:Düşük seviye programlama girişi görevi yapar. RB6:Seri programlama girişi görevi yapar RB7:Seri programlamada data girişi görevi yapar.
RB1	34	G/Ç	TTL	
RB2	35	G/Ç	TTL	
RB3/PGM	36	G/Ç	TTL	
RB4	37	G/Ç	TTL	
RB5	38	G/Ç	TTL	
RB6/PGC	39	G/Ç	TTL/ST	
RB7/PGD	40	G/Ç	TTL/ST	
RC0/T1OSO/T1CKI	15	G/Ç	ST	PORTC: G/Ç olarak yönlendirilebilen port .Pinler G/Ç görevi dışında; RC0:TIMER1 osilatör çıkışı veya TIMER1 clock çıkışı görevi de yapar. RC1:TIMER1 osilatör girişi veya Capture2-G/Compare2-O/PWM2-Ç görevi yapar. RC2:Capture1-G/Compare1-Ç/PWM1-Ç görevi de yapar. RC3: SPI ve I2C modunda senkron seri clock G/Ç görevi yapar. RC4: SPI modunda SPI data giriş, I2C modunda data G/Ç görevi yapar. RC5: SPI modunda SPI data çıkış görevi yapar. RC6:USART asenkron gönderme veya senkron clock görevi yapar. RC7: USART asenkron alma ve senkron data görevi yapar.
RC1/T1OSI/CCP2	16	G/Ç	ST	
RC2/CCP1	17	G/Ç	ST	
RC3/SCK/SCL	18	G/Ç	ST	
RC4/SDI/SDA	23	G/Ç	ST	
RC5/SDO	24	G/Ç	ST	
RC6/TX/CK	25	G/Ç	ST	
RC7/RX/DT	26	G/Ç	ST	
RD0/PSP0	19	G/Ç	ST/TTL	PORTD:G/Ç olarak yönlendirilebilir port veya mikroişlemci hattında arabirim olarak kullanıldığında paralel slave port.
RD1/PSP1	20	G/Ç	ST/TTL	
RD2/PSP2	21	G/Ç	ST/TTL	
RD3/PSP3	22	G/Ç	ST/TTL	
RD4/PSP4	27	G/Ç	ST/TTL	
RD5/PSP5	28	G/Ç	ST/TTL	
RD6/PSP6	29	G/Ç	ST/TTL	
RD7/PSP7	30	G/Ç	ST/TTL	
RE0/RD/AN5	8	G/Ç	ST/TTL	PORTE: G/Ç olarak yönlendirilebilir port . Pinler G/Ç görevi dışında; RE0:Paralel Slave porttan okuma kontrolu veya 5. analog giriş görevi yapar. RE1:Paralel Slave porttan yazma kontrolu veya 6. analog giriş görevi yapar. RE2:Paralel Slave porttan seçim kontrolu veya 7. analog giriş görevi yapar.
RE1/WR/AN6	9	G/Ç	ST/TTL	
RE2/CS/AN7	10	G/Ç	ST/TTL	
VSS	12,31	P	—	Mikrodenetleyici için toprak seviyesini oluşturur.
VDD	11,32	P	—	Mikrodenetleyici için pozitif kaynak gerilimini oluşturur.

RA0, RA1, RA2, RA3, RA4, RA5 bitleri analog/sayısal dönüştürücü olarak ayarlanabilir. A portunu TRISA yazmacı yönlendirir (Giriş ya da çıkış olarak düzenler). Her bir biti bağımsız olarak giriş veya çıkış olarak tanımlanabilmektedir. A portunda

bulunan RA4 biti diğer A portu bacaklarından farklıdır. Bu pin açık kolektör özelliğine sahip olduğu için pull-up direnci ile kullanılmalıdır.

2.2.2 Port B

Port B giriş/çıkış olarak kullanılabilen 8 bitlik bir porttur. Bu portu TRISB yazmacı yönlendirmektedir (Giriş ya da çıkış olarak düzenler). Her bir biti bağımsız olarak giriş veya çıkış olarak tanımlanabilmektedir. Bu port mikrodenetleyici içinde dâhili olarak pull-up yapılmış gibidir. B portunun her bacağı dahili bir dirençle VDD'ye bağlıdır. Bu özellik varsayılan olarak etkin değildir. Fakat OPTION yazmacının 7. Bitini sıfıra çekerek B portunun bu özelliği etkinleştirilebilir. Bu özelliği ile bu port giriş olarak kullanıldığında girişte sinyal yokken giriş lojik-1 olarak algılanır. Bu yapısı ile buton girişi olarak kullanıma uygun bir porttur.

2.2.3 Port C

C portu 8 bitlik sayısal giriş/çıkış portudur. Bu portu TRISC yazmacı yönlendirmektedir (Giriş ya da çıkış olarak düzenler). Her bir biti bağımsız olarak giriş veya çıkış olarak tanımlanabilmektedir. Diğer portlara oranla değişik işlevler için kullanılabilir. Sayısal giriş/çıkıştan başka seçilebilecek fonskiyonlar şunlardır: Timer1 osilatör çıkışı, PWM çıkışları, SPI modu, I^2C modu, USART.

2.2.4 Port D

D portu da C portu gibi 8 bitlik sayısal giriş/çıkış portudur. Bu portu TRISD yazmacı yönlendirmektedir (Giriş ya da çıkış olarak düzenler). Her bir biti bağımsız olarak giriş veya çıkış olarak tanımlanabilmektedir. Tüm port bacakları Schmitt Trigger girişlidir. TRISE yazmacının 4. biti olan PSPMODE bitini 1 yaparak "parallel slave mode"da kullanılabilir. Bu fonksiyon aracılığıyla 8 bit genişliğindeki herhangi bir mikroişlemci bus'ına bağlanabilir.

2.2.5 Port E

E portu 3 bitlik sayısal giriş/çıkış portudur. E portunun kontrolü TRISE tarafından yapılır. Her bir biti bağımsız olarak giriş veya çıkış olarak tanımlanabilmektedir. RE0, RE1 ve RE2 bacaklarında Schmitt Trigger giriş tamponları vardır. Her bir bacak A/S dönüştürücü olarak konfigüre edilebilmektedir. Eğer D portu paralel slave port olarak konfigüre edilirse RE0, RE1, RE2 bacakları D portunun bağlandığı mikrodenetleyici bus'ına sırasıyla READ, WRITE ve CHIP SELECT kontrol girişleri olarak kullanılabilmektedir. Bunun için TRISE uygun biçimde ayarlanmalıdır.

2.3 PIC 16F877'nin Özel Fonksiyonları

2.3.1 USART

USART, yani çok amaçlı senkron/asenkron alıcı/verici PICF877'deki iki seri giriş/çıkış biriminden biridir. Seri iletişim arayüzü (SCI) olarak da bilinen USART, monitör veya PC gibi aygıtlara tam çift yönlü asenkron bağlantıda kullanılmak üzere konfigüre edilmiştir. A/S veya S/A ara yüzlerine, seri kullanılmak üzere konfigüre edilebilmektedir. USART aşağıdaki gibi konfigüre edilebilmektedir.

- Asenkron : Tam çift yönlü,
- Senkron Master : Yarım çift yönlü
- Senkron Slave : Yarım çift yönlü RC6 verici, RC7 ise alıcı port olarak kullanılmaktadır. RCSTA (0x18) ve TXSTA (0x98) yazmaçları da konfigürasyonda kullanılmaktadır [7, 8].

2.3.2 Ana senkronizasyonlu seri (MSSP) port

MSSP (Master Synchronous Serial Port) birimi diğer çevre birimleri ya da mikrodenetleyicilerle haberleşmek için kullanışlı bir modüldür. Bu çevre birimleri seri EEPROM, kaydırmalı yazmaçlar (shift register), gösterge sürücüleri, A/S çeviriciler vb. olabilir. MSSP birimi aynı anda aşağıdaki iki moddan birine konfigüre edilebilir [4].

- Seri çevre ara yüzleri (Serial Peripheral Interface, SPI)
- Bütünleşik devreler (Inter-Integrated Circuit, I^2C)

2.3.3 Analog/Sayısal dönüştürücü birimi

Analog/Sayısal dönüştürücü birimi 10 bittir. Toplamda 8 adet analog/sayısal kanal bulunmaktadır. PIC16F877'de, mikrodenetleyici uyku modunda iken bile analog/sayısal dönüştürücü birimi geri planda çalışmaya devam etmektedir. Analog/Sayısal kanalları için RA4 hariç diğer RA portları ve RE portları kullanılabilir. İlgili yazmaçlar ve adresleri aşağıdaki gibidir [7, 8].

- ADRESH 0x1E; analog/sayısal sonuç yazmacı (high register)
- ADRESl 0x9E; analog/sayısal sonuç yazmacı (low register)
- ADCON0 0x1F; analog/sayısal kontrol yazmacı 0
- ADCON1 0x9F; analog/sayısal kontrol yazmacı 1

BÖLÜM III

MICROBASIC PROGRAMLAMA DİLİ

Bu tez çalışmasında gerçekleştirilen CAN-Bus temelli dağıtık kontrol uygulamasında Microchip (www.microchip.com) firmasının ürettiği mikrodenetleyicilerin ve CAN modüllerinin birlikte kullanımını mümkün kılan Mikroelectronika (www.mikroe.com) firmasının MicroBasic programlama dili kullanılmıştır. Bu dil vasıtasıyla yoğun ve karmaşık CAN yapısı özel komutların verdiği kolaylıkla gerçekleştirilmiştir. Bu bölümde MicroBasic Programlama dili özet olarak incelenecektir.

MicroBasic programı PROGRAM kelimesi ile başlar. Bu kelimeden sonra programa herhangi bir isim verilebilir. “END.” komutu ile program bitirilmelidir. Bu komut derleyiciye programın bittiğini belirtir ve aynı zamanda programın sonunda sonsuz bir döngü oluşturur (programın o noktada durması için). Daha karmaşık MicroBasic programlarında semboller, sabit sayılar ve değişkenler kullanılabilir [9]. Eğer; sembol, değişken ve sabit sayılar kullanılmışsa, şu sıraya göre yazılmalıdır:

- Programın başında PROGRAM kelimesi
- Sembol kullanılmışsa SYMBOL listesi
- Sabit kullanılmışsa CONST listesi
- Değişken kullanılmışsa DIM listesi
- Alt Prosedür (Sub Procedure) ve Alt Fonksiyon (Sub Function) Program yazılımı
- Programın sonunda END. kelimesi

3.1 Temel Program Yapısı

Bir MicroBasic programının içermesi muhtemel tüm bileşenler Şekil 3.1'de görülmektedir.

```
Program (program ismi)
'tanımlar (globaller)
Include        (dahil edilen diğer birimler)
'sembol tanımları
const …
'değişken tanımlar
dim …
'yordam tanımları
sub procedure altyordam_ismi (...)
        {yerel tanımlar}
end sub
'fonksiyon tanımları
sub function fonksiyon_ismi(…)
…
end sub
'program gövdesi
main :
'kodunuzu buraya yazınız .
end.
```

Şekil 3.1 MicroBasic program yapısı [9]

3.2 Modüller (Modules)

MicroBasic'te, her proje bir tek proje dosyası ile birlikte bir veya daha fazla modül dosyası içermektedir. Proje dosyası, .pbp uzantılı olup proje hakkında bilgi içerir, modül dosyası ise .pbas uzantılıdır ve gerçek kaynak kodlarını içerir. Modüller şunları yapabilmeyi sağlar:

- Bağımsız olarak düzenlenebilecek şekilde uzun programları daha küçük parçalara ayırmayı,
- Farklı projelerde kullanılabilecek kütüphaneler oluşturmayı,
- Kaynak kodunu saklı tutarak, kütüphanelerden diğer program geliştiricilerinin yararlanmasını sağlar.

Birimler kendi dosyalarında kaydedilmişlerdir ve ayrı ayrı derlenirler. Derlenen birim oluşturulan bir uygulamaya bağlanır. Bir proje oluşturmak için, derleyicinin bir kaynak dosyasına ya da her birim için derlenmiş birim dosyasına ihtiyacı vardır [9].

3.3 İçerik Cümleciği (Include Clause)

MicroBasic birimleri, “include” cümleciği ile tanımlanır. Rezerve edilmiş include kelimesini ve çift tırnak içinde birim ismini belirlemek gerekir. Dosya uzantısı kullanılmamalıdır. Include cümleciğine sadece bir dosya dahil edilebilir. Ancak kaynak dosyasının içerisinde birden fazla include cümleciği olabilir. Fakat hemen program (veya birim) isminden sonra belirtilmelidirler [9]. Aşağıda örnek birim tanımlamaları görülmektedir.

```
Örnek:
Program Projem
Include ‘‘birim’’
Include ‘‘alfasayisal’’
…
```

Bir birim ismi verildiğinde, derleyici .mcl ve .pbas dosyalarının bulunup bulunmadığını belirten arama yollarını takip ederek kontrol eder.

Eğer .mcl ve .pbas dosyaları bulunursa, derleyici onların tarihlerini kontrol eder ve daha yeni olanı projeye kaydeder. Eğer .pbas dosyası .mcl’den daha yeni ise yeni kütüphane eskisinin üzerine kaydedilecektir. Eğer sadece .pbas dosyası bulunursa, derleyici yeni bir .mcl dosyasını oluşturacak ve onu projeye dahil edecektir. Eğer sadece .mcl dosyası bulunursa, yani hiçbir kaynak kod yok ise, derleyici dosyayı bulduğu şekilde kapsar. Eğer her iki dosya da bulunmazsa derleyici ‘‘File not found’’uyarısı verecektir [9].

3.4 Ana Birim

MicroBasic'te her projenin tek bir ana birime ihtiyacı vardır. Ana birim kodun hemen başında program anahtar kelimesi ile tanımlanır ve derleyicinin nereden başlaması gerektiğini gösterir.

Proje sihirbazı (Project Wizard) ile boş bir proje oluşturulduğunda, Kod Editör'ü yeni bir ana birim gösterecektir ki bu programın sade halidir [9]. Aşağıda Projem adında örnek bir proje görülmektedir. Görülen kısım en sade projedir.

```
Program Projem
        'sabitler, değişkenler, etiketler, alt program ve fonksiyonları buraya yazılır
main:
        'program kodunuzu buraya yerleştiriniz
end.
```

Program anahtar kelimesinden önce sadece kullanıcı yorumları bulunabilir. Program isminden sonra ihtiyaca göre include cümlecikleri koyulabilir. Tüm genel değişken tanımları (sabitler, değişkenler, etiketler, alt program ve fonksiyonlar) main etiketinden önce yerleştirilmelidir. MicroBasic'te, son komut "END." her programın sonunda bulunur ve sonsuz döngü işlevi görür.

3.5 Diğer Birimler

"Main" ile başlayan ana birimin haricinde, "Module" anahtar kelimesi ile başlayan birimlerdir. Yeni oluşturulan boş birim yalın haldedir. Aşağıda boş birim görülmektedir.

```
Module Mymodule
        Implements
End
```

Yorumlar hariç hiçbir şey "Module" anahtar kelimesinden önce gelmemelidir. Birim isminden sonra, isteğe bağlı olarak bir veya birden fazla include cümleciği yerleştirilebilir.

3.6 Arabirim Bölümü

Birimin “implements” anahtar kelimesinin üstünde kalan kısmı arabirim bölümüdür. Buraya proje için gerekli genel tanımlar yerleştirilebilir (sabitler, değişkenler, etiketler v.b).

Arabirim bölümünde prosedürler tanımlanamaz. Onun yerine, birimin dışında da görmek istenilen prosedürlerin prototipleri belirtilebilir. Prototipler tanımlamalarla tamamen uyuşmalıdır [9].

3.7 Değişkenler (Variables)

Değişkenler, değerleri program çalışırken değişebilen nesnelerdir. Her değişkenin geçerli bir tanımlayıcıya sahip bir ismi olması gerekmektedir. Bu isim değişkenin saklı olduğu hafıza yerleşkesine ulaşmak için kullanılır.

Değişkenler dosya veya yordamın tanım bölümlerinde tanımlanmalıdır, her değişken kullanılmadan önce tanımlanmış olmalıdır. Genel değişkenler, yani bir bloğa özgü olmayan değişkenler, include cümleciğinin altında ve ana program main anahtar kelimesinin üstünde tanımlanmalıdır.

Her değişken için bir veri tipi belirtmek zorunludur. MicroBasic değişkenler için aşağıda gösterilen sözdizimini kullanır:

```
DIM değişken_adları as değişken tipi
```

Burada, değişken_adları; virgüllerle ayrılmış geçerli tanımlayıcılar listesidir. Tip herhangi bir veri tipi olabilir. Aşağıda değişken tanımlaması ile ilgili birkaç örnek verilmiştir.

```
DIM i, j, k as byte
DIM sayici, deger as word
```

3.8 Değişken Tipleri

Byte, **char** ve **word** değişkenler işaretsiz değişkenlerdir ve birçok mikrodenetleyici tabanlı projelerde işaretsiz değişkenler kullanılmaktadır. **Short**, **integer**, **longint** değişkenleri ise işaretli değişkenlerdir [9]. Çizelge 3.1'de değişken tipleri görülmektedir.

Çizelge 3.1 Değişken tipleri

DEĞİŞKEN	BİT SAYISI	SAYI BÜYÜKLÜĞÜ
Byte	8	0 … 255
Char	8	0 … 255
Word	16	0 … 65535
Short	8	-128….127
Integer	16	-32768… 32767
Longint	32	-2147483648… 2147483647
Float	32	$\pm 1.17549535082*10^{-38}$….. $\pm 6.90564774407*10^{38}$

3.9 Değişkenler ve PIC

Her tanımlı değişken RAM bellekte bir bölüm işgal eder. Değişkenin veri tipi sadece temsil edeceği değerin erişim aralığını da belirler. Şunu unutmamak lazım ki, farklı değişken tipleri kullanmak program işleme zamanını da değiştirecektir. MicroBasic yerel değişkenlerin RAM bellekte işgal ettiği bölgeyi düzenler, farklı fonksiyonlarda ve prosedürler tanımlanmış yerel değişkenleri mümkün olduğu sürece aynı bellek bölgesine yerleştirir [9].

PIC'in özel fonksiyon kayıtlarını ayrıca tanımlamaya gerek yoktur. Zira MicroBasic otomatik olarak uygun kaydedici byte tipindeki genel değişkenlere eşler. Örneğin T0IE, INTF v.b.

3.10 Sabitler (Constants)

Sabitler, değerleri program süresince değişmeyen verilerdir. Bir programda sabit kullanma RAM belleğinden tasarrufu sağlar. Sabitler her ifadede kullanılabilir ama onlara yeni bir değer atanamaz [9].

Sabitler program veya yordamın tanım bölümünde tanımlanmalıdırlar. “Conts” anahtar kelimesinden sonra istenildiği kadar sabit tanımlanabilir.

```
Const sabit_ismi [as tipi] = deger
```

Her sabit tek sabit_ismi’ne sahiptir ve bu isim geçerli bir tanımlayıcı olmalıdır. Sabit isimlerinin büyük harfle yazılması gelenek haline gelmiştir. Sabit değer için atama bekleyecektir ki bu değer belirlenen tip için geçerli olmalıdır. Tip tanımı ihtiyaridir, tip belirtilmemişse derleyici değeri barındırabilecek en küçük tipi atar. Eğer bir sabit dizisi (array) tanımlaniyorsa, tipi belirlemek gerekir [9]. Aşağıda sabit tanımlama örnekleri verilmiştir.

```
conts MAKSIMUM as longint = 10000
conts MINIMUM = 1000            ‘derleyici word tipi atayacaktır
conts ANAHTAR = ’’n’’           ‘derleyici char tipi atayacaktır
conts MSG = “Merhaba”           ’derleyici string tipi atayacaktır
conts AYLAR as byte [12]=(31,28,31,30,31,30,31,31,30,31,30,31)
```

3.11 Etiketler (Labels)

Etiketler, goto ve gosub komutlarının hedefleri olarak kullanılırlar. İstenilen komuta uygun bir etiket “:” karekteri ile aşağıdaki gibi atanabilir. Aşağıda etiket tanımlama biçimi gösterilmiştir.

```
Etiket_adi : ifade
```

MicroBasic’te etiket için özel bir tanımlama gerekli değildir. Etiket ismi geçerli bir tanımlayıcı olmalıdır. Etiketli komut ve goto/gosup komutu aynı blok içinde bulunmalıdır. Dolayısıyla başka bir prosedür veya fonksiyona dallanma söz konusu

olamaz. Bir blok içinde aynı etiket birden fazla komuta atanamaz. “Main” etiketi bir programın başlangıç noktasını işaretler ve her projenin ana biriminde bulunmalıdır [9].

Aşağıda basla alt yordamını sürekli çağıran bir sonsuz döngü örneği gösterilmiştir.

```
kal: basla
goto kal
```

3.12 Semboller (Symbols)

BASIC sembolleri parametre kullanmadan basit makrolar oluşturmayı sağlar. Böylece herhangi bir kod satırı tek bir tanımlayıcı takma ada eşlenebilir. Semboller, uygun kullanılırsa, program kodunun okunaklılığını ve yeniden kullanılabilirliğini arttırırlar.

Semboller birimin hemen başında birim isminden ve include cümleciğinden hemen sonra tanımlanmalıdır. Bir sembolün kapsamı (scope) her zaman içinde tanımlanmış olduğu dosya ile sınırlıdır [9]. Sembol şöylece belirtilir:

```
symbol alias = code
```

Burada, takma ad (alias) program kodu içinde sürekli kullanılacak geçerli bir tanımlayıcı olmalıdır. Bu tanımlayıcının da kapsamı dosyadır. Code herhangi bir komut satırı olabilir (atamalar, fonksiyon çağırmalar v.b). Sembol kullanımı RAM bellekten hiç yer kullanmaz, derleyici basitçe sembolü gördüğü her yere eşlenik kod satırını yazar. Aşağıda bazı örnekler görülmektedir.

```
symbol deger = 155 ‘numerik değer için takma ad
symbol PORT = PORTC ‘SFR için takma ad
dim cnt as byte ‘bazı değişken
‘…
main :
if cnt > deger then
cnt = 0
PORT.1 = 0
End if
```

3.13 Kullanılmayan Değişken İsimleri

Çizelge 3.2'de verilen değişken isimleri MicroBasic derleyici tarafından kullanıldığından bu değişken isimleri kullanılmamalıdır.

Çizelge 3.2 Kullanılamayan değişken isimleri

Absolute	abs	array	asm	begin	boolean
Case	char	chr	clear	const	dim
Div	do	double	else	end	exit
Float	for	function	goto	gosub	if
Include	in	int	integer	interrupt	is
Loop	label	mod	module	message	new
Next	not	or	org	print	procedure
Program	read	select	sub	step	string
Switch	then	to	until	wend	while
With	xor				

3.14 Operatörler (Operators)

Operatörler, bir ifadenin, değişken ve diğer nesnelerine uygulandığında bazı hesaplamaları tetikleyen sembollerdir. MicroBasic'te dört tip operatör mevcuttur:

- Aritmetik Operatörler (Arithmetic Operators)
- Bit-işlem Operatörleri
- Boole (Boolean) Operatörleri
- İlişkili (Relational) Operatörler

3.14.1 Operatörlerin önceliği ve birleşmesi

MicroBasic'te dört tane öncelik kategorisi mevcuttur. Öncelik değeri yükseldikçe işlem sırası öne gelmektedir. Aynı kategorideki operatörler eşit önceliğe sahiptirler.

Her kategori bir birleşme kuralına sahiptir. Sağdan sola veya soldan sağa parantezlerin yokluğunda, bu kurallar eşit öncelikli operatör içeren ifadeleri gruplarlar. Çizelge 3.3'te operatör öncelikleri verilmiştir.

Çizelge 3.3 Operatör öncelikleri

Öncelik	İşlenen	Operatör	Birleşme
4	1	@ not + -	sağdan sola
3	2	* / div mod and shl shr	soldan sağa
2	2	+ - or xor	soldan sağa
1	2	= <> < > <= >=	soldan sağa

3.15 Aritmetik Operatörler

Aritmetik operatörler matematiksel hesaplamaları gerçekleştirmek için kullanılırlar. Sayısal işlenenler üzerinde çalışırlar ve sayısal sonuçlar üretirler. Char operatörleri teknik olarak byte olduklarından aritmetik işlemlerde işaretsiz olarak kullanılabilirler. Tüm aritmetik operatörler soldan sağa doğru birleştirilirler [9]. Çizelge 3.4'te aritmetik operatörler ve öncelik değerleri görülmektedir.

Çizelge 3.4 Aritmetik operatörler

Operatör	İşlem	Öncelik
+	Toplama	2
-	Çıkarma	2
*	Çarpma	3
/	Bölme	3
div	Bölme, aşağıya en yakın tamsayıya yuvarlanır	3
mod	Tamsayı bölmenin kalanını döndürür(kayan noktalı sayılarla kullanılmaz)	3

(-) operatörü, bir işaretli değerin işaretini değiştirmek için tekli bir ön ek operatörü gibi kullanılabilir. (+) tekli ön ek operatörü de bu işlev için kullanılabilir ama veriyi etkilemez. Örnek olarak:

```
b = -a
```

3.16 İlişkisel (Relational) Operatörler

İlişkisel operatörler, ifadelerin eşit olup olmadığını test etmek için kullanılırlar. Tüm ilişkisel operatörler DOĞRU (TRUE) veya YANLIŞ (FALSE) sonucunu verirler. Tüm ilişkisel operatörler soldan sağa doğru bir araya gelirler [9]. Çizelge 3.5'te ilişkisel operatörler ve öncelik değerleri görülmektedir.

Çizelge 3.5 İfadelerdeki ilişkisel operatörler

Operatör	İşlem	Öncelik
=	eşit	1
<>	eşit değil	1
>	büyük	1
<	küçük	1
>=	büyük veya eşit	1
<=	küçük veya eşit	1

3.17 Bit-İşlem (Bitwise) Operatörü

Bit-işlem operatörü sayısal bir işlenenin her bitini tek tek değiştirebilmek amacıyla kullanılır. Bu operatörlerin kullanımında ya her iki işlenen de işaretli ya da her ikisi de işaretsiz olmalıdır [9].

Bit-işlem operatörleri soldan sağa doğru birleşir. Sadece kural dışı olarak, tümleyici bit-işlem operatörü soldan sağa doğru birleşmez. Çizelge 3.6'da bit-işlem operatörleri ve öncelik değerleri görülmektedir.

Çizelge 3.6 Bit-işlem operatörleri

Operatör	İşlem	Öncelik
and	Bit-işlem VE ; eğer her iki bit de 1 ise,1, diğer durumlarda 0 alır.	3
or	Bit-işlem VEYA ; bitlerden biri veya her ikisi 1 ise 1, diğer durumda 0 alır.	2
xor	Exclusive VEYA (XOR) ; eğer bitler tümleyici ise 1, diğer durumda 0 alır.	2
not	Her bitin evriğini alır.	4
shl	Bitleri sola kaydırır	3
shr	Bitleri sağa kaydırır	3

Bit-işlem operatörlerinden and, or ve xor kendi işlenenlerinin bitleri üzerinde mantıksal işlem gerçekleştirirler. Not operatörü işlenenin her bitinin evriğini alır.

3.18 İşaretsiz Dönüştürme

Eğer sayı az karmaşıktan çok karmaşık veri tipine çevriliyorsa, üst byte'lar sıfırlarla doldurulur. Eğer çok karmaşıktan az karmaşık veri tipine çevriliyorsa veri kırpılır (üst byte'lar kaybolur). Örnek olarak :

```
dim a as byte dim b as word
...
a = $AA
b = $F0F0
b = b and a
```

a, sıfırlarla uzatılır dolayısıyla b, $00A0 olur.

3.19 İşaretli Dönüştürme

Sayı az karmaşık veri tipinden çok karmaşık veri tipine çevriliyorsa ve eğer işaret biti 1(negatif sayı) ise üst bitler 1 ile doldurulur; eğer işaret biti 0 ise (pozitif sayı) üst bitler sıfırlarla doldurulur. Eğer çok karmaşıktan az karmaşık veri tipine çevriliyorsa veri kırpılır (üst byte'lar kaybolur) [9]. Örnek olarak :

```
dim a as byte dim b as word
...
a = -12
b = $70FF
b = b and a
```

a, işaretli olarak uzatılır, dolayısıyla üst bitler $FF ve b ise $70F4 olur.

3.20 Bit-İşlem Kaydırma Operatörleri

İkilik (binary) operatörlerden "<<" ve ">>" sol işlenenin bitlerini, sağ işlenen tarafından belirlenen kadar, sola veya sağa kaydırırlar. Sağ işlenen pozitif ve 255'den küçük olmalıdır [9].

Sola kaydırma ile (<<), en soldaki bitler düşer ve yeni bitler sağ taraftan 0 olarak atanır. Dolayısıyla işaretsiz bir byte veriyi sola n tane kadırmak veriyi 2n ile çarpmak demektir. Bu işlem, kaydırma işleminde düşen bitler işaret biti ise, işaretli veriler için de geçerlidir.

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

Örneğin üst byte'ı ayrıştırmak istenirse; işlem şu şekilde gerçekleştirilebilir:

```
PORTB = word (temp >> 8)
```

3.21 İfadeler (Expressions)

Bir ifade (expression) sonuç üreten operatör, işlenen ve noktalama işaret dizisidir. Birincil ifadeler; sabitler, değişkenler ve fonksiyon çağırmaları içerir. Operatörler ve birincil ifadeleri kullanarak daha karmaşık ifadeler oluşturulabilir. Formal olarak ifadeler tekrarlanarak tanımlanırlar; alt ifadeler bellek sınırına kadar iç içe yuvalanabilir. İfadeler, parantezin varlığına, kullanılan operatörlere ve işlenenlerin veri tipine bağlı olarak dönüşüm, gruplama, birleşim ve öncelik kurallarına tabi tutulurlar. Operatörlerin ve alt ifadelerin gruplama şekilleri, bunların MicroBasic'teki gerçek gruplama şekillerini göstermeyebilir. Atama ifadelerinde veya aritmetik ve mantık operatörlerinin bulunduğu ifadelerde işaretli ve işaretsiz veri tipleri karıştırılamaz, ancak açık dönüşümler yapılabilir.

BÖLÜM IV

CAN-BUS PROTOKOLÜ

Küçük boyut, düşük maliyet, yüksek güvenilirlik ve yüksek hız gibi özellikleri, CAN teknolojisinin endüstriyel uygulamalarda yaygın olarak kullanımına yol açmıştır. Önceleri otomobil üzerindeki tüm elektronik kontrol sistemlerini tek bir hat üzerinden haberleştirmek amacıyla kullanılmasına rağmen, uygulama kapsamı hızla gelişerek, PLC kontrollü üretim sistemleri, robotik, tıbbi aletler, gemi otomasyon sistemleri, uzay uçuş simülatörleri gibi farklı alanları da içerisine almıştır [10, 11]. Dağıtık kontrol uygulamalarında da yaygın olarak kullanılan bir haberleşme türü olmuştur. Bu bölümde CAN-Bus protokolü ile ilgili bir inceleme sunulmuştur.

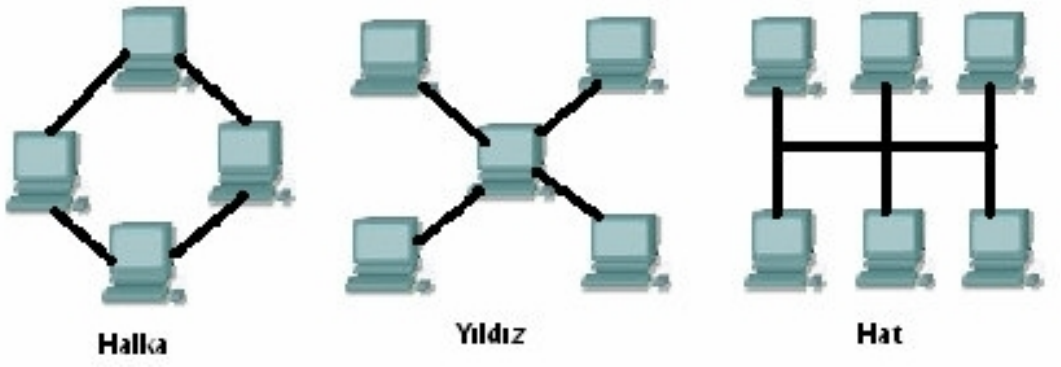
4.1 Giriş

CAN, otomotiv sektöründe kullanılmak üzere tasarlanmış, yüksek verimli bir ileri asenkron seri iletişim protokolüdür. Açılımı “Controller Area Network Bus”olan yani “Kontrol Alan Ağı Veri yolu”dur. 1980’lerde Robert Bosch tarafından otomotivde kablo yumağı yerine bir kablodan yazılım kontrollü veri transferini sağlamak amacıyla geliştirilmiştir. CAN, otomotiv endüstrisindeki en çok bilinen haberleşme sistemidir. CAN standart protokolü, OSI (Open System Interconnect – Açık Sistem Bağlantısı) referans modelinin 1. ve 2. katmanlarına karşılık gelmektedir. Her ne kadar başlangıçta yalnızca otomotiv uygulamaları için tasarlanmış olsa da küçük boyut, düşük maliyet, yüksek güvenilirlik ve yüksek hız gibi özelliklerinden dolayı birçok dağıtık endüstriyel kontrol uygulamalarında yaygın olarak kullanılmaktadır. Güvenliğin çok önemli olduğu gerçek zamanlı uygulamalarda da kullanılır. Öyle ki istatistiksel olasılık hesapları sonucunda bir asırda bir tane tespit edilemeyen mesaj hatası yapabileceği tespit edilmiştir [10, 11].

4.2 Can Topolojisi

CAN topolojisi, network üzerindeki CAN modüllerini birbirine bağlayan kablonun yapısıdır. CAN protokolünde kullanılan topolojiler Hat, Halka veya Yıldız

topolojilerinden biri olabilir. Şekil 4.1'de CAN protokolünde kullanılan ağ topolojileri görülmektedir.



Şekil 4.1 CAN protokolünde kullanılan topolojiler

CAN Protokolünde kullanılan en yaygın topoloji tipi, Hat topolojisidir [17]. Bu topolojide, her modül ağ içerisindeki mesajlaşmayı sağlayabilmek için tek bir seri kablo ile diğer modüllere bağlıdır. İstasyonlar arasında CSMA/CD+CR erişim kuralları kullanılır. Bu kurallar, iki modül aynı anda mesaj gönderdiğinde ortaya çıkan çakışmaları engellemek için kullanılır [12, 15].

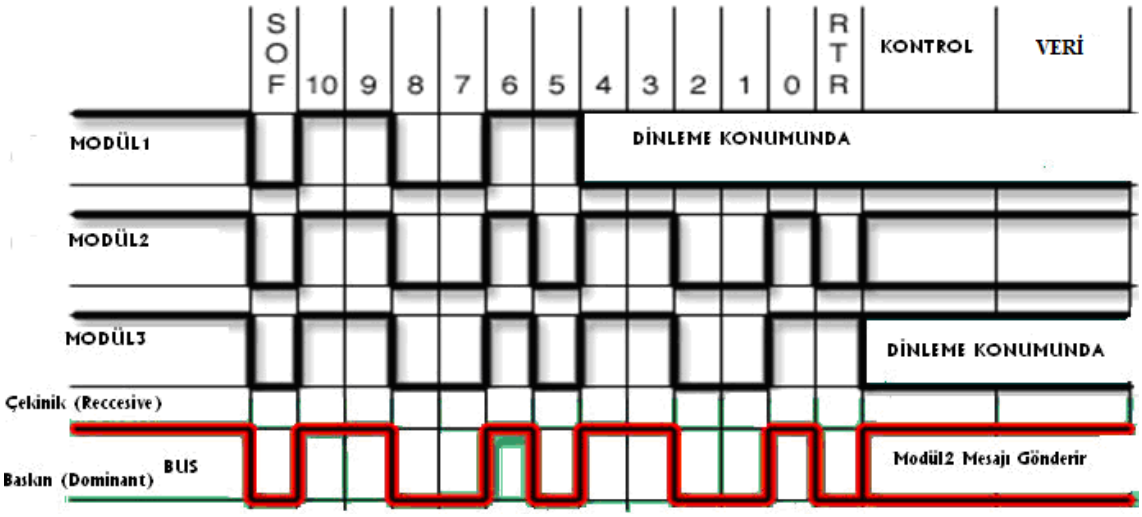
4.3. Çoklu Taşıyıcı Giriş Duyusu (Carrier Sense Multiple Access - CSMA)

CAN protokolü, CSMA/CD+CR (Carrier Sense Multiple Access With Collision Detection And Collision Resolution) yapısında bir protokoldür. CSMA (Carrier Sense Multiple Access), 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 (Collision Detection) ise, ağda iki modülün aynı zamanda mesaj göndermeye başlaması neticesinde oluşan çakışmanın algılanmasıdır. CR (Collision Resolution) ise, bu çakışmanın çözümlenmesi anlamına gelmektedir. Çakışmanın çözümlenmesinde kullanılan yöntem ise uzlaştırma (arbitration) yöntemidir [18, 19].

CAN protokolü, bit seviyesinde bozucu olmayan uzlaştırma metodunu kullanmaktadır. Bunun anlamı, bir çakışma olayı meydana geldiğinde uzlaştırma tamamlandıktan sonra mesajın bozulmadan, orijinal haliyle kalabilmesidir. Uzlaştırma işlemi neticesinde,

yüksek önceliğe sahip olan mesaj gecikmeksizin ve bozulmaksızın gönderilebilmektedir [12].

Bit seviyesinde uzlaştırma işleminin gerçekleşebilmesi için iki önemli noktanın oluşması gerekmektedir. Birincisi, sistemdeki lojik durumların baskın (dominant) ve çekinik (recessive) olarak tanımlanması gerekir [20]. İkincisi ise, bus üzerinde gerçek 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 uzlaştırma işlemini kazanacaktır. CAN mesaj çerçevesi (Frame) içerisinde bulunan mesaj belirleyicideki (Message Identifier - uzlaştırma 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 gerçek lojik durumu anlayabilmeleri için bus'ı izleyeceklerdir. Düşük öncelikli mesaj, bir noktadan sonra çekinik bir bit göndermeyi deneyecektir. Bu noktada, bu biti gönderen modül uzlaştırma işlemini kaybedecek ve aniden mesaj gönderme işlemini durduracaktır. Yüksek önceliğe sahip mesaj, gönderme işlemi bitene kadar bus üzerinde istenen yerlere gidecektir [21]. Uzlaştırma işlemini kaybeden modül bir sonraki boş işlem periyodunu bekleyecek ve daha sonra mesajını tekrar gönderecektir. Bu durum Şekil 4.2'de gösterilmiştir.



Şekil 4.2 Uzlaştırma (Arbitration) işlemi

Şekilde gerçekleşen işlem kısaca şu şekilde açıklanabilir. Modül1 5 nolu işlemde çekinik olan tanımlandığından dinleme konumuna geçmiştir. Devamında ise Modül2 ve Modül3 baskın olduklarından uzlaştırma işlemine tabi tutulmuşlardır. Daha sonra gelen RTR (Remote Transmit Request – Endirekt İleti Talebi) değerinde data verisi olduğunu gösteren ve baskın olan Modül2, mesajı göndermekte ve Modül3 dinleme konumuna geçmektedir. Modül2 mesajını ilettikten sonra Modül3 ve akabinde de Modül1 verilerini gönderebilecektir.

4.4 Mesaj Tabanlı Haberleşme

CAN protokolü, mesaj tabanlı bir protokoldür. Mesaj gönderme işlemi modüllerin adreslerine göre yapılmamaktadır [22]. 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 [23]. Bir mesaj, sistemin tasarım şekline göre ya belirli bir modüle ya da birçok modüle gönderilebilir [18, 19].

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 [12, 23].

4.5 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 maddi 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

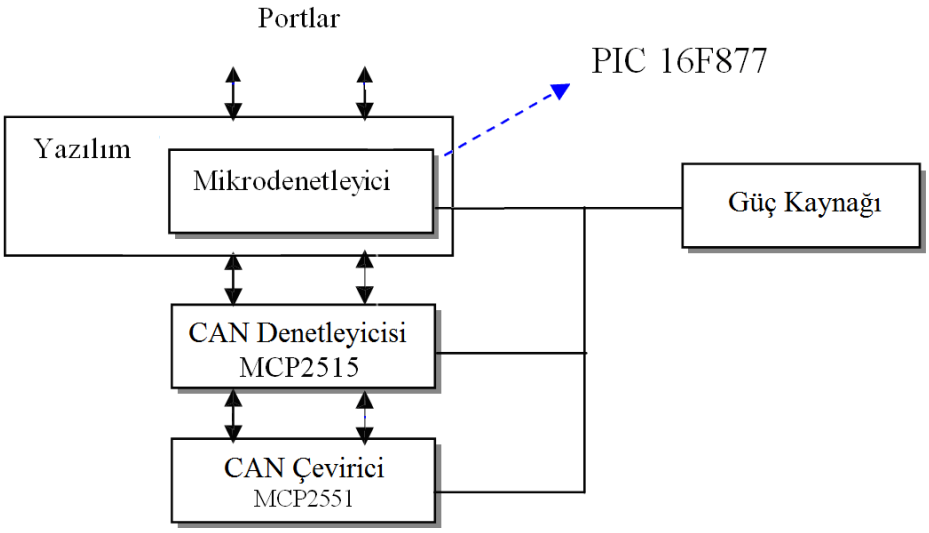
tek bir tip kablo olmalıdır [24]. CAN protokolü maksimum 1 Mbit/sec haberleşme hızına sahiptir. Tabii ki bu hız farklı iletim mesafelerine göre değişebilmektedir. Çizelge 4.1'de iletim mesafelerine göre bus hızları görülmektedir.

Çizelge 4.1 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
1000	40 Kbit/s

4.6 Can Modülü

CAN Modülü, CAN hattı üzerinden gelen mesajlara göre hareket eden elektronik yapıdır. 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 4.3'te basit bir CAN modülü görülmektedir.



Şekil 4.3 CAN Modülü

Sistemde kullanılan elektronik bir modül içerisinde aşağıdaki yapılar bulunur:

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

Fiziksel ara yüz, bus üzerindeki elektriksel sinyali lojik seviyeye çevirmek için kullanılır. Bu eleman yardımıyla, CAN denetleyicisi ve fiziksel bus arasında bir bağlantı sağlanır. CAN entegresi olarak bilinen bus denetleyicisi, kullanıcı bilgilerini CAN mesaj çerçevesi içerisinde iletim kanalı üzerinden gönderilen fiziksel bitlere çevirir [25]. Piyasada 3 çeşit CAN entegresi bulunmaktadır [12]. Bunlar;

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

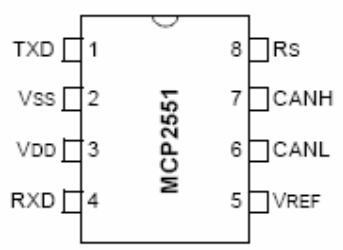
Stand–Alone entegreler, data ve adres hatları üzerinden sistemi yöneten mikro denetleyiciye bağlanırlar. Bütünleşik yapıda olan entegreler içerisinde hem sistemi yöneten mikrodenetleyici hem de CAN kontrolünü yapan CAN modülü bulunmaktadır. Serial–Linked tipi entegreler yaygın olarak kullanılmazlar [22]. Bu tip entegrelerin programlanabilmesi için sistem içerisinde bir başka mikrodenetleyiciye ihtiyaç vardır [12].

Mikrodenetleyici üzerindeki yazılım, kullanıcı tanımlı uygulamaları çalıştırmak için kullanılmaktadır. Portlar ve CAN entegresi arasındaki veri yakalama işlemini kontrol eder. Veri yakalama işlemi süresince, mesajların tamponlanma metoduna göre CAN denetleyicileri iki sınıfa ayrılmaktadır. Eğer mesajlar CAN denetleyicisinin hafızasında tutulacaksa buna FullCan adı verilir. Eğer mesaj mikrodenetleyicinin belleğinde tutulacak ise buna BasicCan adı verilir [22]. BasicCan, mesaj yakalama işleminde FullCan'e göre daha esnek ve hızlıdır [12].

4.6.1 MCP 2551

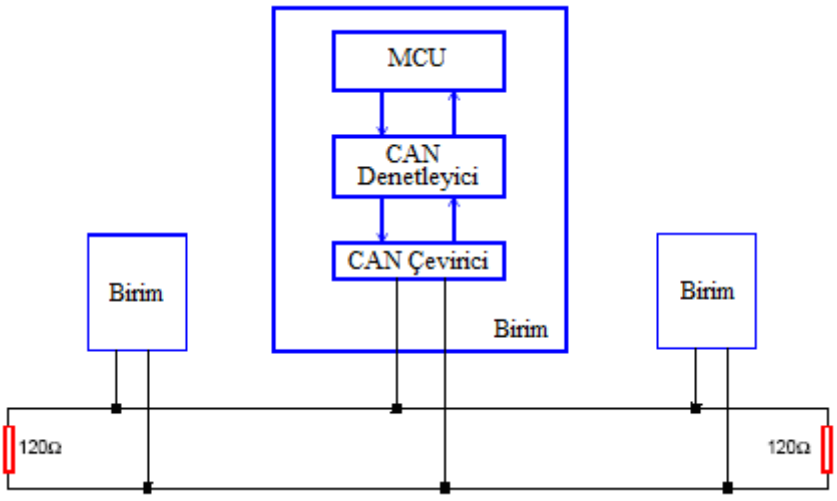
CAN modülü içerisinde yer alan MCP2551 entegresi Microchip firması tarafından üretilmektedir.

MCP2551 entegresi, CAN-Bus üzerinden gelen mesajları CAN modülüne, CAN modülünden gelen mesajları CAN-Bus'a aktaran yüksek hızlı CAN alıcı vericisidir. Şekil 4.4'te MCP2551 entegrenin bacak bağlantıları gösterilmektedir [25].



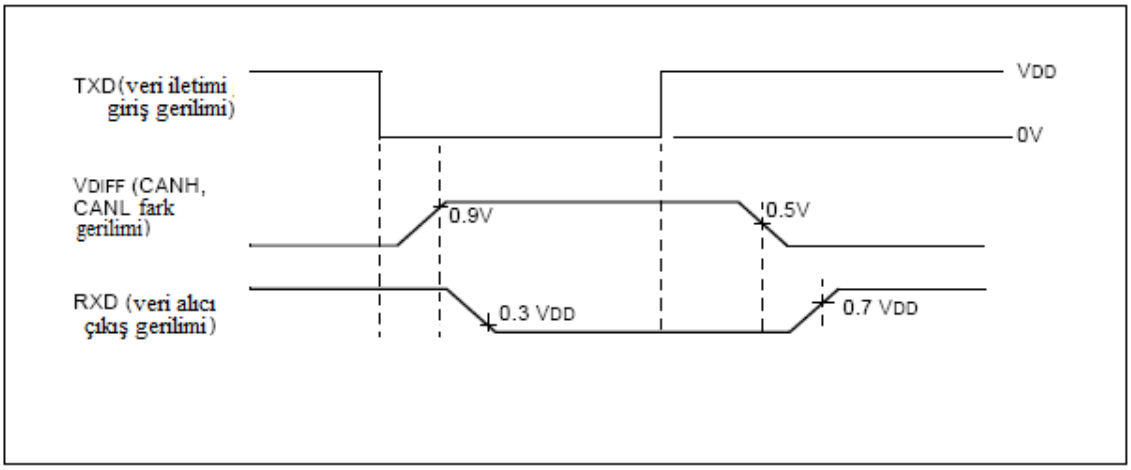
Şekil 4.4 MCP2551 entegre bacak bağlantıları

CAN haberleşme sistemi içerisinde bulunan her bir modül, CAN-Bus kablosu üzerine yükleyeceği dijital sinyalleri, iletim için uygun bir seviyeye çekmesi gerekir. MCP2551 entegresi, CAN modülü ile bus üzerinde dış kaynaklardan dolayı oluşan yüksek gerilim dalgalarını tamponlama özelliğine sahiptir. Mikrodenetleyicinin CAN-Bus üzerine bağlantısı Şekil 4.5'te görülmektedir.

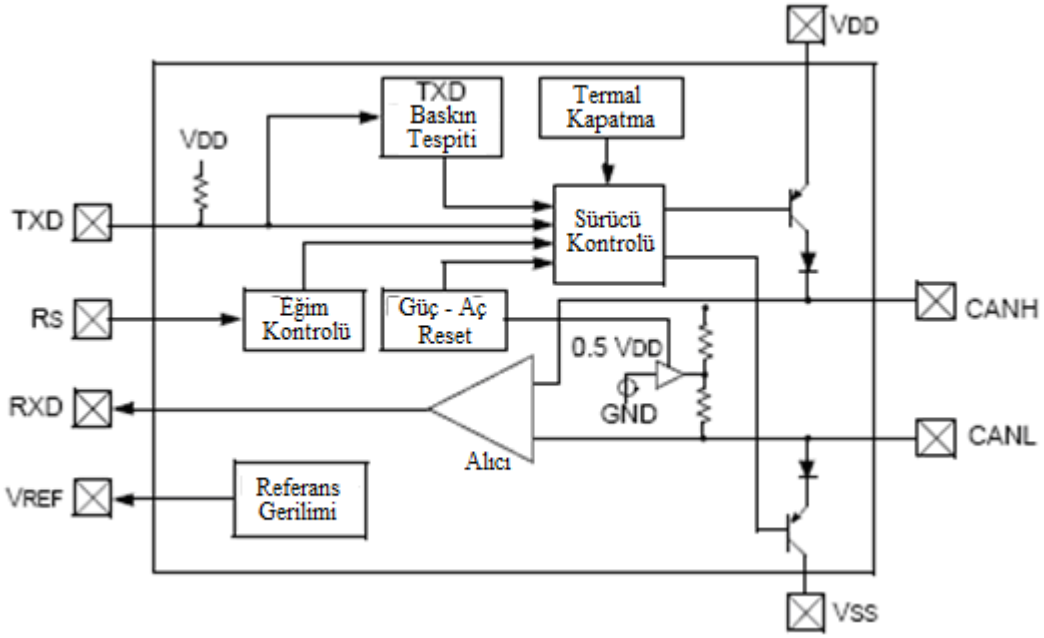


Şekil 4.5 Mikrodenetleyicinin CAN-Bus üzerine bağlantısı

CAN-Bus üzerinde 2 tane lojik seviye bulunur. Bunlar baskın (lojik 0) ve çekinik (lojik 1) gerilim seviyeleridir. Dominant lojik seviye, CANH ve CANL arasındaki fark gerilimi, tanımlı olan gerilim seviyesinin üzerinde olduğu zaman oluşur [26]. Tanımlı olan seviye, 1.2V civarındadır. Çekinik lojik durum ise, iki hat arasındaki fark geriliminin tanımlı olan seviyenin altında olduğu zaman oluşur. Genelde bu seviye 0V'tur. Şekil 4.6'da CANH ve CANL arasındaki gerilim seviyeleri görülmektedir.



Şekil 4.6 CAN Bus lojik gerilim seviyeleri [25]

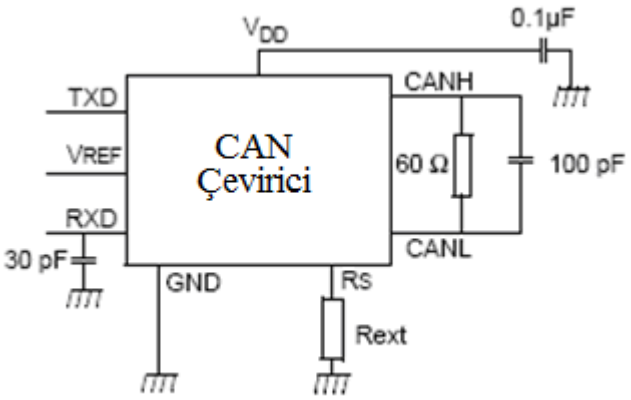


Şekil 4.7 MCP2551 entegresinin blok diyagramı

MCP2551 entegresinin blok diyagramı Şekil 4.7'deki gibidir.MCP2551, CAN hattından aldığı verileri RX formatında iletmektedir. TX formatında aldığı verileri CAN hattına iletmektedir.

Microchip firmasının ürettiği PIC mikrodenetleyicilerinin bazılarının içerisinde CAN modülü yer almaktadır. Fakat bu modül içerisinde CAN çeviricisi yer almamaktadır. Bu nedenle CAN işlemlerinde çevirici kullanılması şarttır. Bu entegre ile içerisinde CAN modülü bulunan herhangi bir mikrodenetleyici ile CAN çevirici olarak kullanılabilir.

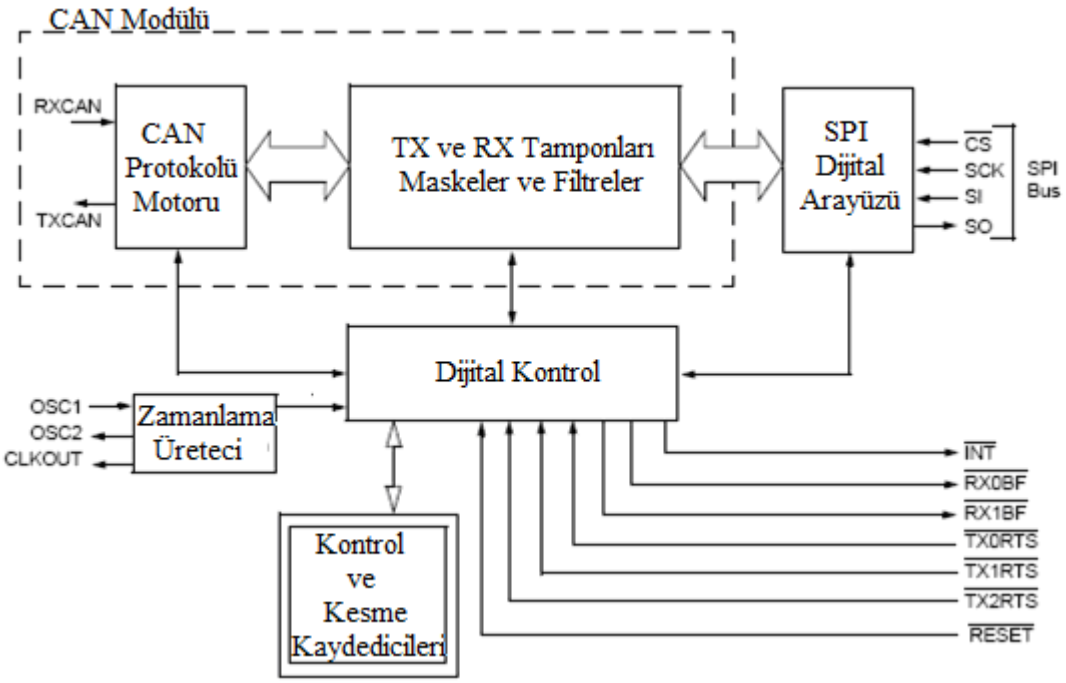
MCP25512in çalışması için gerekli olan devre şeması Şekil 4.8'de görülmektedir.



Şekil 4.8 MCP2551 bağlantı şeması

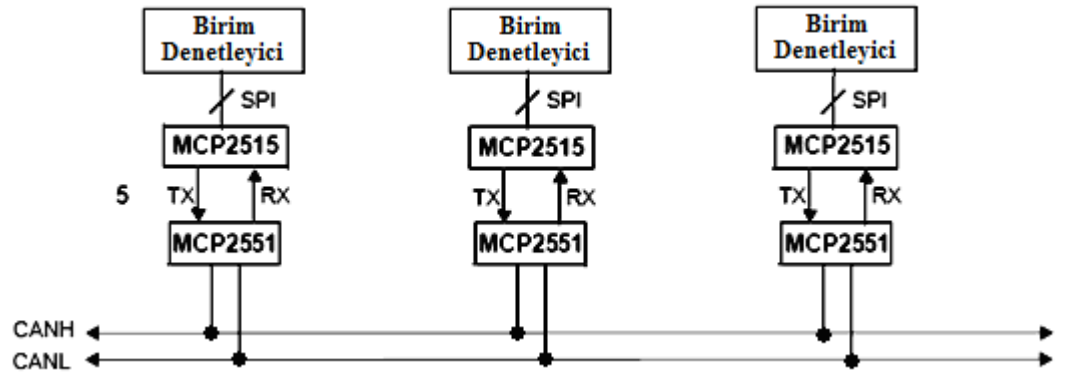
4.6.2 MCP 2515

Oluşturulacak CAN modülü içerinde yer alan MCP2515 entegresi Microchip firması tarafından üretilmektedir. Bu entegre ile içerisinde CAN modülü olmayan, mikrodenetleyicilere CAN haberleşmesi sağlanmaktadır. Seri çevresel arayüz (Serial Peripheral Interface, SPI) birimi olarak tasarlanmıştır. SPI olarak tasarlanan bu entegre, devrelere ve donanımlara fonksiyonellik, yenilenme ve değişebilirlik katmaktadır. Şekil 4.9'da entegrenin blok diyagramı görülmektedir.



Şekil 4.9 MCP2515'in blok diyagramı

CAN çevirici modülle senkron çalışmaktadır. Mikrodenetleyici ile seri çevresel arayüzler (Serial Peripheral Interface, SPI) ile haberleşmektedir. Bu haberleşmeye uygun bir Mikrodenetleyici kullanılması zorunluluğu unutulmamalıdır.



Şekil 4.10 Örnek bağlantı şeması

Örnek bir bağlantı şeması Şekil 4.10'da verilmiştir. Bu şekilde de görüldüğü gibi CAN hattındaki sinyaller çevirici vasıtasıyla TX-RX formatına alınmakta ve SPI yardımıyla mikrodenetleyici ile haberleşmektedir.

4.7 CAN Mesaj Çerçeveleri

CAN protokolü içerisinde 4 farklı mesaj tipi bulunur [19]. Bunlar;

- Veri Çerçevesi (Data Frame)
- Uzak Çerçeve (Remote Frame)
- Hata Çerçevesi (Error Frame)
- Taşma Çerçevesi (Overload Frame)

Bu mesaj çerçevelerinden, veri çerçevesi içerisinde veri, diğerlerinde ise kontrol amaçlı mesajlar bulunur.

4.7.1 Veri çerçevesi (Data Frame)

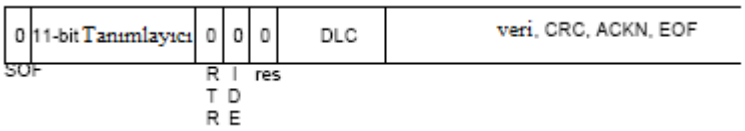
CAN tarafından tanımlanmış olan mesaj hakkında ekli bilgi sağlayan alanları içerir. İki tip veri çerçevesi bulunur [12]. Bunlar;

- Standart Veri Çerçevesi (Standart Data Frame)
- Genişletilmiş Veri Çerçevesi (Extended Data Frame)

Her iki çerçeve içerisinde ortak olarak farklı amaçlar için kullanılan yedi adet alan bulunmaktadır [12]. Bunlar;

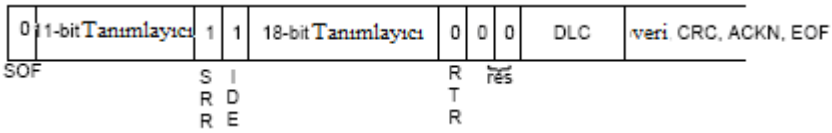
- Başlangıç Biti (Start of Frame)
- Uzlaştırma Alanı (Arbitration Field)
- Kontrol Alanı (Control Field)
- Veri Alanı (Data Field)
- CRC (Cyclic Redundancy Check) Alanı (Cyclic Redundancy Check Field)
- ACK (Acknowledge) Alanı (Acknowledge Field)
- Çerçeve Sonu (End of Frame)

Standart çerçeve içerisinde 12 bitlik uzlaştırma alanı bulunur. Bu 12 bitten 11'i mesajı tanımlamak için, 1 bit ise RTR (Remote Transmit Request) bitidir. Bu bit veri çerçevesinin uzaktaki bir modülden istekte bulunabilmesi için kullanılır. Şekil 4.11'de standart bir veri çerçevesi görülmektedir [27].



Şekil 4.11 Standart veri çerçevesi

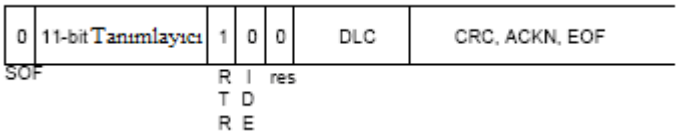
Genişletilmiş veri çerçevesi içerisinde 32 bitlik uzlaştırma alanı bulunur. 32 bitin 29'u mesajı tanımlamak için kullanılır.1 bit IDE (Identifier-Belirleyici) bitidir. Bu bit veri çerçevesinin genişletilmiş formatta olduğunu göstermek için kullanılır. 1 bit SRR biti bulunur. Bu bit kullanılmaz. 1 bit de RTR (Remote Transmit Request – uzak ileti talebi) biti bulunur. Bu bit çerçevenin remote ya da veri olduğunu göstermek için kullanılır [27]. Şekil 4.12'de genişletilmiş veri çerçevesi görülmektedir.



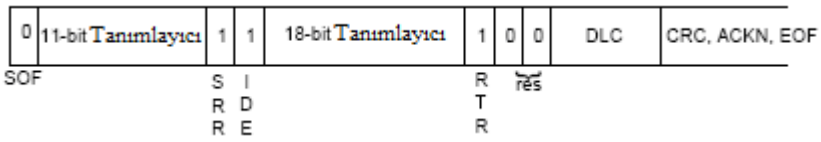
Şekil 4.12 Genişletilmiş veri çerçevesi

4.7.2 Uzak çerçeve (Remote Frame)

Veri çerçevesi içerisinde bulunan RTR biti, bir modülün başka bir modülden bilgi istemesi durumunda kullanılmaktadır. Uzak çerçeve içerisinde veri alanı bulunmamaktadır. Bir uzak çerçevenin yapısı, standart çerçeve için Şekil 4.13'te ve genişletilmiş çerçeve için Şekil 4.14'te görülmektedir. Bir modül, uzak çerçeve gönderirken istekte bulunduğu veri çerçevesinin tanımlayıcı bilgisini koymalı ve RTR bitini lojik 1 yaparak göndermesi gerekmektedir [18]. Hem veri hem de uzak çerçevelerin uzlaştırma alanında bulunan RTR biti, uzak çerçeve ile veri çerçevesi arasındaki farkı oluşturmaktadır. Eğer RTR biti lojik 1 ise bu çerçevenin uzak çerçeve, lojik 0 ise veri çerçevesi olduğu anlaşılır [12].



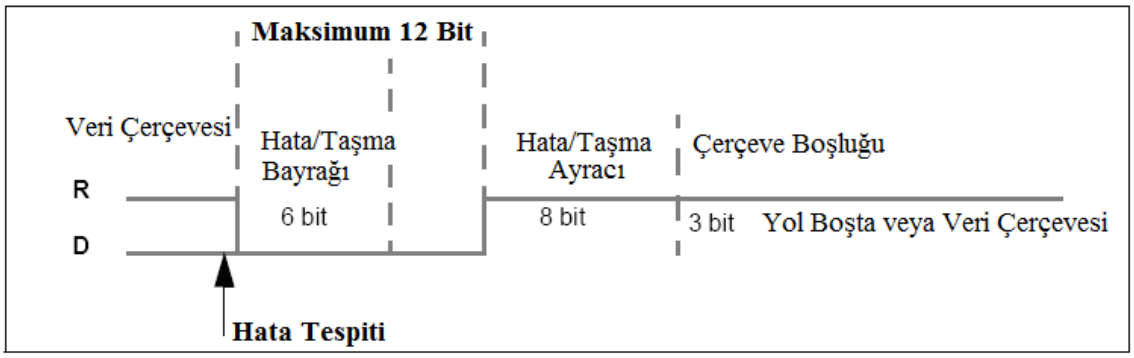
Şekil 4.13 Standart uzak çerçeve



Şekil 4.14 Genişletilmiş uzak çerçeve

4.7.3 Hata çerçevesi (Error Frame)

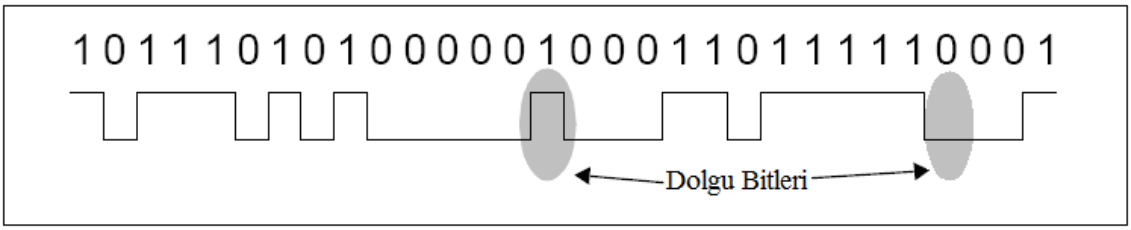
Bu çerçeve, bir mesaj gönderildikten sonra veya gönderim süresince bir hata oluştuğu zaman gönderilen kontrol amaçlı bir çerçevedir. Şekil 15'te hata çerçevesinin yapısı görülmektedir. Bir hata çerçevesi içerisinde 6 veya 12 adet aynı lojik seviyeli bit ve 8 adet hata sınırlama biti bulunmaktadır. Her aynı seviyeli 6 bitlik grup hata bayrağı olarak adlandırılmaktadır. Gönderici ya da alıcı modül hata bayrağı gönderdiği zaman diğer modüller bit dolgu (bit stuffing) kuralının ihlal edildiğini anlar ve hepsi tekrar hata bayrağı gönderirler.



Şekil 4.15 Hata/Taşma çerçevesi yapısı

Bit dolgu (bit stuffing) metodu, CAN veri ya da uzak çerçevesi içerisinde ardı ardına aynı lojik seviyeli 5 bit gönderildiği zaman 6. bitine ters lojik seviyeli bitin eklenmesidir [28]. Bu ters lojik seviyeli bitin eklenmesinin sebebi, örnekleme işlemi için çok fazla sayıda kenar kullanarak çerçeve senkronizasyonunun sağlanmasıdır. Eklenen bu bit,

alıcı tarafından ardı ardına aynı seviyeli 5 bitin gözlenmesinden sonra 6. bitin silinmesiyle kaldırılır. Şekil 4.16'da bit dolgu işlemi görülmektedir.



Şekil 4.16 Bit dolgu İşlemi

4.7.4 Taşma çerçevesi (Overload Frame)

Taşma çerçevesi, alıcı durumdaki modülün bir sonraki gelecek olan mesaj için dahili çalışma zamanının yeterli olmaması ve çerçeveler arası alanı içerisinde baskın (Lojik 0) bir bitin bulunması durumunda gönderilir [12]. Şekil 4.15'te taşma çerçevesi yapısı görülmektedir.

4.8 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 [12]. Tespit edilen hata yapıları;

- CRC (Cyclic Redundancy Check – periyodik fazlalık testi) Hatası
- Teyit Hatası (Acknowledge Error)
- Şekil Hatası (Form Error)
- Bit Hatası (Bit Error)
- Dolgu Hatası (Stuff Error)

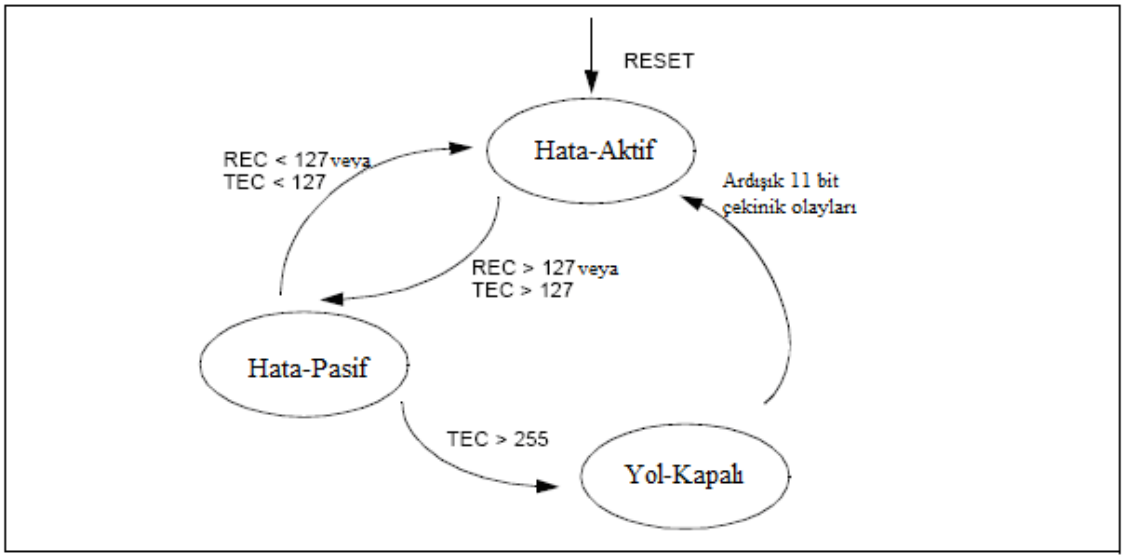
Data frame içerisinde bulunan CRC alanı içerisinde gönderilen mesajın toplam bit sayısı bulunmaktadır. Eğer herhangi bir modül gelen mesajın toplam bit sayısı ile CRC alanındaki sayı arasında farklılık görürse CRC hatası oluşur. Veri çerçevesi içerisinde

bulunan ACK Slot biti mesaj gönderilirken 1 yapılır. Eğer herhangi bir modül bu biti 0 olarak görürse teyit hatası oluşur. Eğer herhangi bir modül CAN protokolünün içerisinde kullanılan mesaj çerçevelerinin yapısında bir bozukluk tespit ederse şekil hatası oluşur [15]. Eğer gönderici modül lojik 1 biti gönderirken bus üzerinde o biti lojik 0 olarak görürse ya da lojik 0 biti gönderirken bus üzerinde o biti lojik 1 olarak görürse bit hatası oluşur. Eğer mesaj çerçevesi içerisinde aynı seviyeli 6 bit tespit edilirse dolgu hatası oluşur.

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

- Hata – Aktif (Error – Active)
- Hata – Pasif (Error – Passive)
- Hat – Kapalı (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 (Receiver Error Counter) sayacıdır. TEC, mesaj gönderilirken oluşan hataları sayar, REC ise mesaj alınırken oluşan hataları sayar. Şekil 4.17'de sayaçlar ve hata modları gösterilmektedir.



Şekil 4.17 TEC ve REC sayaçları ve hata modları

Şekil 4.17’de görüldüğü 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.

BÖLÜM V

DÖRT KATLI MODEL ASANSÖR

Bu yüksek lisans çalışmasında dağıtık kontrol uygulaması olarak dört katlı model bir asansör kullanılmıştır. Bilindiği gibi günümüzde anasör devrelerinde yüksek kablolama maliyetinden dolayı çareler aranmaktadır. Bu kablolama maliyetini en asgariye indirme yollarından biri CAN-Bus sistemi kullanılmaktadır. Bu nedenle CAN-Bus haberleşme sistemi model asansöre uygulanmıştır. Bu bölümde uygulamada kullanılacak olan dört katlı model asansör incelenmektedir.

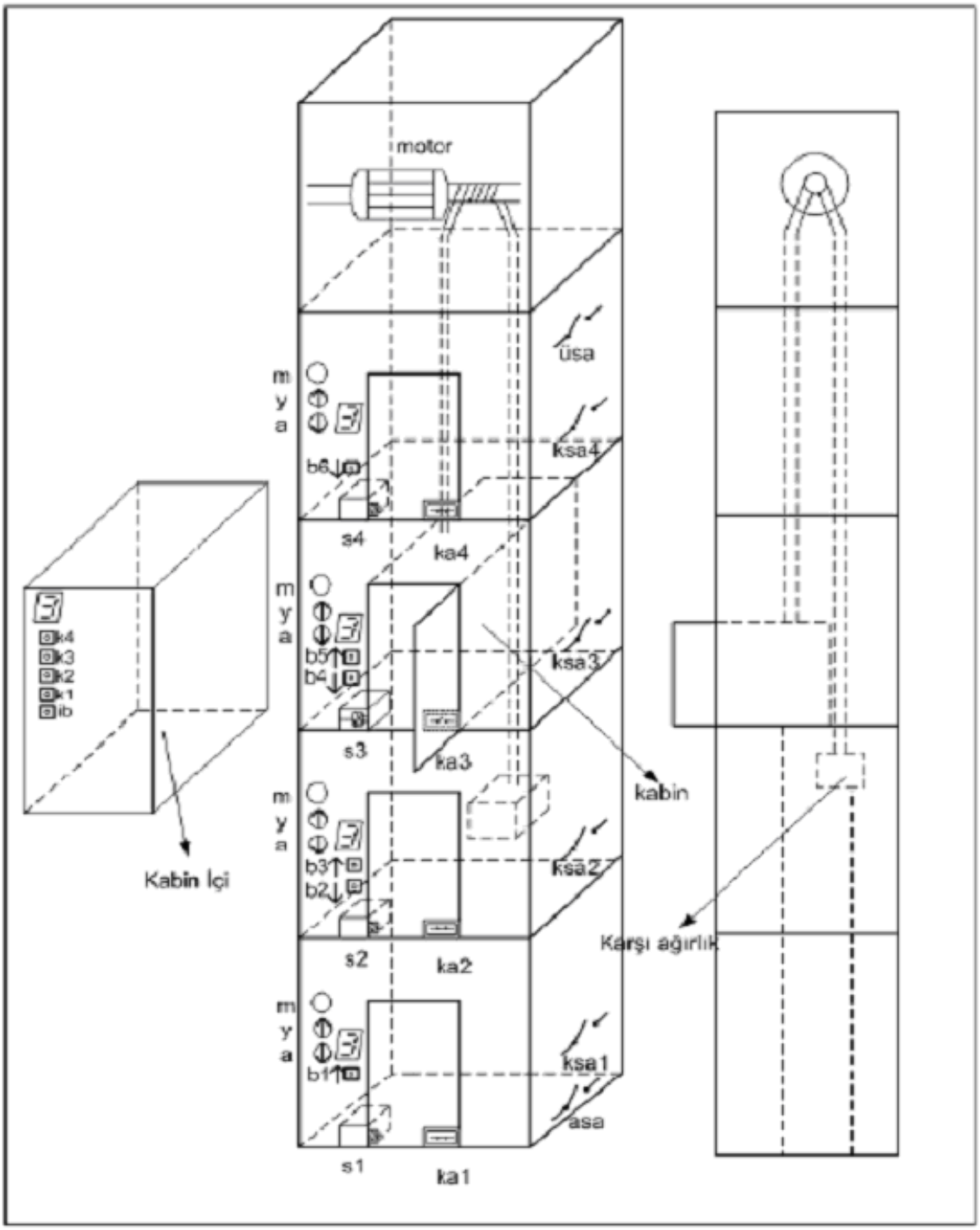
5.1 Model Asansör

Kullanılan model asansör dört kattan oluşmaktadır. Şekil 5.1'de model asansörün şekli ve bu modelde kullanılan elemanlar, bu elemanlara verilen semboller görülmektedir.

Her katta kapı güvenliğini sağlamak amacıyla kullanılan **selenoid valfler** bulunmaktadır. Bu selenoid valfler enerjilenmediğinde itili konumda kalarak kapıyı kilitli tutmakta ve kabin kat hizasında değilken kapının açılmasına müsaade etmemekte, kabin kat hizasına geldiğinde ise enerjilenip konum değiştirerek (yani çekili konuma geçerek) kapının açılmasına müsaade edip kabine girilmesine ya da kabinden çıkılmasına imkan tanımaktadır. Katlarda bulunan bu selenoidler 12V DC ile çalışmaktadır.

Her katta bir **kat sınır anahtarı** bulunmaktadır. Bu anahtarlar kabinin kat hizasına gelip gelmediğini anlamak için kullanılmıştır. Bu anahtarlar normalde açık kontağa sahip olup kabin kat hizasına geldiğinde açık kontak kapanarak kabinin katta olduğu sinyalini verir. Kabin kata geldiğinde kat sınır anahtarından +5V DC, katta değilken 0V alınmaktadır.

Her katın kapısında **kapı anahtarları** bulunur. Kapılarda bulunan kapı anahtarları sayesinde kapının açık mı kapalı mı olduğu anlaşılır. Kapı açık iken 0V, kapalı iken +5V DC alınmaktadır [13].



Şekil 5.1 Dört katlı model asansör

Şekil 5.1'de görülen dört katlı model asansörde, kullanılan elemanlara verilen sembollerin tanımları Çizelge 5.1'de görülmektedir.

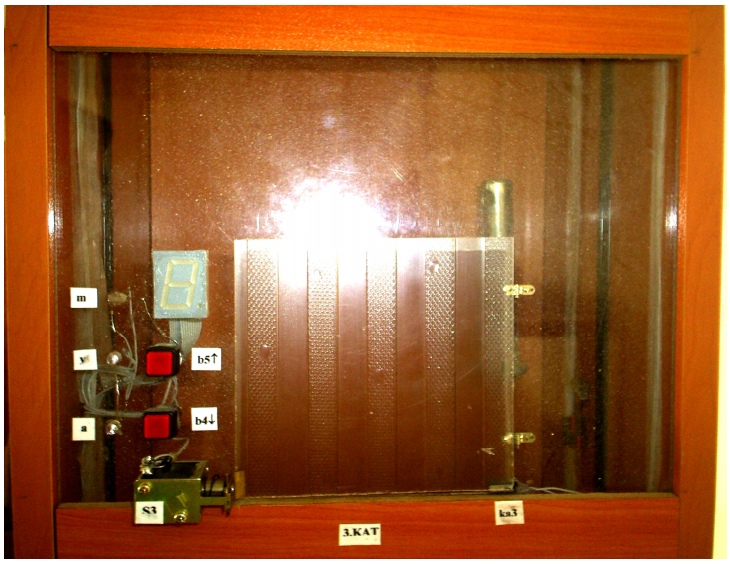
En alt katta **alt sınır anahtarı** kullanılmıştır. Bu anahtar; kabinin bu seviyenin altına inmesini engellemek için kullanılmıştır. Alt sınır anahtarı da normalde açık bir kontağa

sahip olup kabin bu sınıra ulaştığında açık kontak kapanacaktır. Kabin bu seviyeye geldiğinde alt sınır anahtarından +5V DC diğer durumlarda 0V alınmaktadır.

Çizelge 5.1 Model asansörde kullanılan sembollerin tanımları

SEMBOLLER	TANIMLARI
ka1, ka2, ka3 ve ka4	Kapı anahtarları
s1, s2, s3 ve s4	Selenoid valfler
b2, b4 ve b6	Katlardaki aşağı yön butonları
b1, b3 ve b5	Katlardaki yukarı yön butonları
ksa1, ksa2, ksa3 ve ksa4	Kat sınır anahtarları
k1, k2, k3, k4	Kabin içindeki butonlar
asa	Alt sınır anahtarı
üsa	Üst sınır anahtarı
m	Meşgul lambası
a	Kabin aşağı gidiyor lambası
y	Kabin yukarı gidiyor lambası
ib	İmdat butonu

Bu model asansörün bir katında kullanılan elemanlar, Fotoğraf 5.1'de görülmektedir.



Fotoğraf 5.1 Model asansörün 3. katında kullanılan elemanlar

En üst katta **üst sınır anahtarı** kullanılmıştır. Bu anahtar; alt sınır anahtarı ile benzer mantıkta çalışmaktadır. Yani kabinin bu seviyenin üstüne çıkmasına engel olmak maksadıyla kullanılmıştır. Kabin bu seviyeye geldiğinde üst sınır anahtarından +5V DC diğer durumlarda 0V alınmaktadır.

Her katta **meşgul lambası** kullanılmıştır. Bu lamba (LED) kabin hareket halindeyken ya da kapılardan birisi açıkken yani kabine biniliyor ya da iniliyorken katlara kabinin kullanılmakta olduğu sinyalini iletmektedir.

Her katta **kabin aşağı yönde gidiyor** ya da **kabin yukarı yönde gidiyor lambaları** (LED'leri) kullanılmıştır. Bu lambalar kabinin hangi yönde hareket ettiği sinyalini katlara iletmektedir.

Her katta **kabin çağrı butonları** bulunmaktadır. Katlardan kabini çağırmak ve hangi yönde gidileceği sinyalini iletmek maksadıyla kullanılmışlardır. Kabin çağrı butonlarına basılı iken +5V DC basılmadığı durumda ise 0V alınmaktadır.

Kabin içerisinde **kat butonları** bulunmaktadır. Bunlar kabin içerisindeki kişilerin kaçıncı kata gitmek istediklerini belirtmeleri maksadıyla kullanılmışlardır. Kat butonlarına basılı iken +5V DC basılmadığı durumda ise 0 V alınmaktadır.

Her katta ve kabin içerisinde **7 parçalı gösterge** (7 segment display**)** kullanılmıştır. Bunlar kabinin kaçıncı katta olduğunu katlara ve kabin içindeki kişilere iletmek maksadıyla kullanılmışlardır.

Yine kabin içerisinde **imdat butonu** bulunmaktadır. Bu buton herhangi bir nedenle asansör kabininin katlar arasında kalınması durumunu, ifade etmek maksadıyla kullanılmıştır.

Model asansörün üstündeki ayrı bir bölümde ise kabinin aşağı veya yukarı yönde hareketini sağlayan bir 12V'luk DC **motor** bulunmaktadır.

Kullanılan dört katlı model asansörün resmi, Fotoğraf 5.2’de görülmektedir.

Fotoğraf 5.2 Dört katlı model asansörün görünümü

BÖLÜM VI

CAN-BUS İLE GERÇEKLEŞTİRİLEN DAĞITIK KONTROL UYGULAMASI

Dağıtık Kontrol Sistemleri (DKS) temel olarak üç bileşenden oluşurlar. Bunlar, merkez birim (Master Station), uzak uç birim (Remote Terminal Unit) ve iletişim birimleridir [14].

Uzak uç birim, DKS'nin izlenmesi ve kumanda edilmesi gereken tüm birimlerine yerleştirilen bileşenler olup karar verme mekanizmalarına sahip değildirler. Donanımsal ve yazılımsal bileşenleri kısıtlı olup gerçek zamanlı ve önceden tanımlı işlemleri yürütmek üzere tasarlanmışlardır.

Merkezi birim, işlemci, bellek, disk ve ağ ara yüzlerinden oluşan bilgisayar donanımları ile genel veya özel amaçlı bir işletim sistemi ve onun üzerinde koşan DKS yazılımlarından oluşur [14]. DKS merkezi birim yazılımı genel olarak şu iki fonksiyonu yerine getirmek üzere tasarlanır. Birincisi, uzak uç birimlerinden alarm ve verileri toplayıp işlemek ve bunları raporlamak üzere saklamaktır. Diğer temel fonksiyonu ise uzak uç birimlerinden toplanan verileri değerlendirerek tekrar uzak uç birimlere gönderilmek üzere uygun komutlar üretmektir [14].

İletişim birimleri, merkez ve uzak uç birimler arasındaki haberleşmeyi sağlayan erişim ortamlarını ve haberleşme protokollerini kapsar. DKS'ler için iletişim birimlerinin gerçek zamanlı ve kesintisiz aktarım yapabilme şartlarını her durumda sağlamaları beklenir. İletişim birimleri için alınacak her türlü güvenlik tedbirlerinin bu iki şartı sağlaması önemlidir [14].

Bu çalışmada dört katlı bir model asansör dağıtık kontrol uygulaması örneği olarak ele alınmış ve CAN-Bus protokolü kullanılarak gerçekleştirilmiştir. Bu bölümde gerçekleştirilen bu dağıtık kontrol uygulamasının donanım ve yazılım yapıları detaylı olarak incelenmektedir. Öncelikli olarak sistem donanımı detaylı olarak ele alınmakta, sonrasında MicroBasic programı kullanılarak gerçekleştirilen sistem yazılımı incelenmektedir.

6.1 Donanım

Dağıtık Kontrol Sistemi olarak düşünüldüğünde model asansör için uzak uç birimleri ve merkez biriminin tespiti yapılmıştır. Bu tespite göre katlar ve kabin uzak uç birimi olarak tasarlanmıştır. Merkezi birim olarak da motoru ve sistemi kontrol edecek ayrı bir donanım tasarlanmıştır. Toplam olarak beş uzak uç birim ve bir adet merkezi birim gerçekleştirilmiştir.

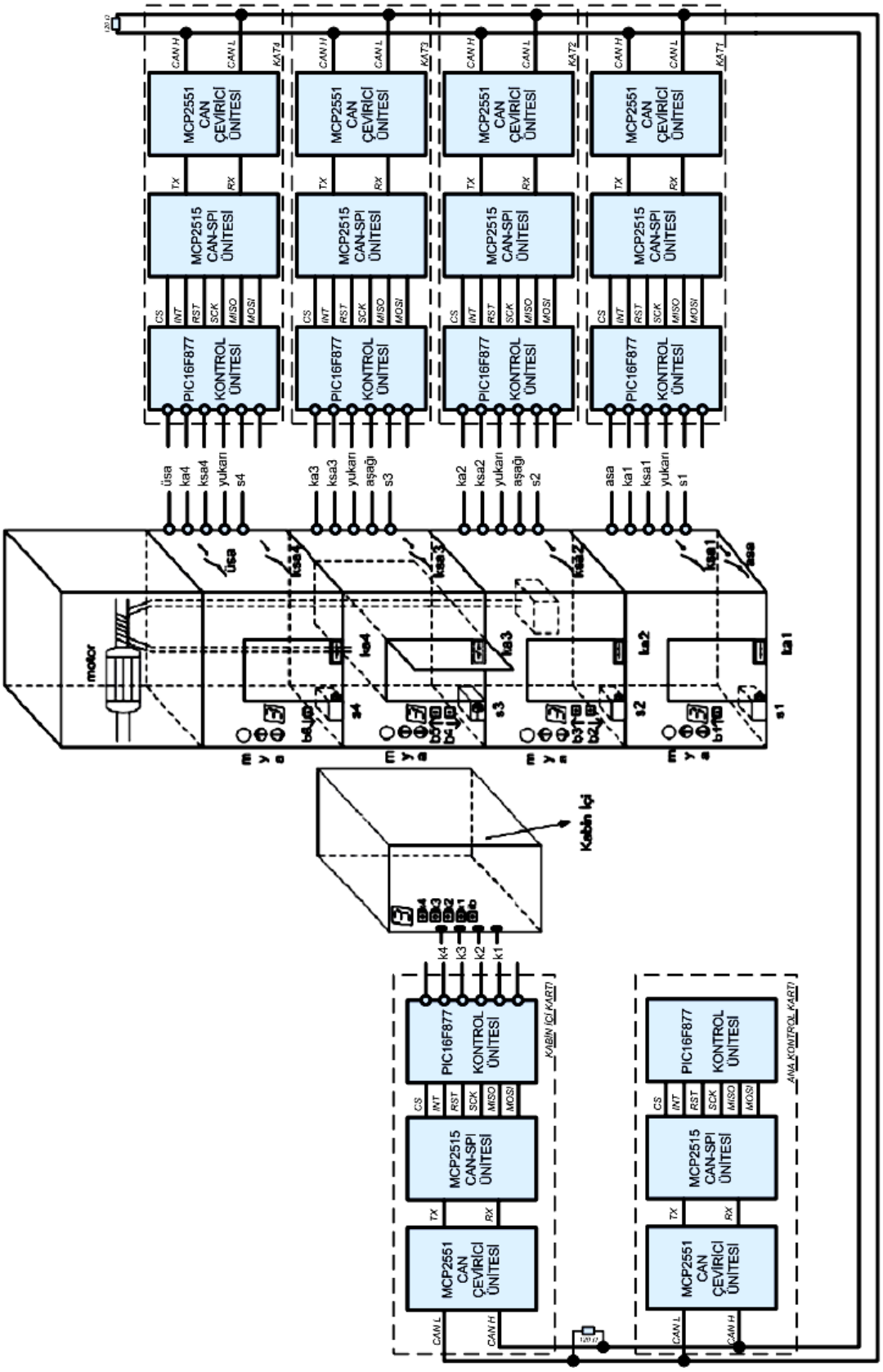
Gerçekleştirilen her bir birim için bir adet PIC16F877 mikrodenetleyicisi, bir MCP2551 CAN-Bus çevirici ünitesi ve bir MCP2515 CAN-SPI ünitesi kullanılmıştır. Şekil 6.1'de oluşturulan dağıtık kontrol sisteminin blok diyagramı görülmektedir. Toplam altı modülün haberleşmesi CAN-Bus protokolü yardımıyla gerçekleştirilmiştir.

Gerçekleştirilen dağıtık kontrol sisteminde, her bir kat için bir kart tasarımı, kabin için bir kart tasarımı ve kontrol ünitesi için bir kart tasarımları gerçekleştirilmiştir. Gerçekleştirilen her bir kartta mikrodenetleyicilerin devreden çıkarılmadan programlanmalarını sağlayacak şekilde bir düzenleme yapılmıştır. Yapılan tasarımlar her bir ünite için aşağıda anlatılmaktadır.

6.1.1 PIC16F877'li kat kontrol kartı tasarımı

Dört katlı model asansör incelendiğinde, ikinci ve üçüncü katların girişlerinin aynı olduğu görülmüştür. İkinci ve üçüncü katlardan farklı olarak birinci katta alt sınır anahtarı, dördüncü katta da üst sınır anahtarı bulunmaktadır. Tüm katlardaki kontrol edilmesi gereken çıkışlar aynıdır.

Detaylı bir inceleme ile katlar için gerçekleştirilecek olan devre tasarımlarının benzer olacağı anlaşılmıştır. Bunun üzerine katlar için her kattaki işlevi yerine getirebilecek özellikte, tek bir tip kart tasarlanması kararlaştırılmıştır.



Şekil 6.1 Dört katlı model asansör devresi için CAN-Bus protokolü ile gerçekleştirilen dağıtık kontrol sistemine ait blok diyagramı

Gerçekleştirilen kartın girişleri aşağıda görülmektedir.

- Kat sınır anahtarı (ksa)
- Kat çağırma butonları (b)
- Kat kapı anahtarı (ka)
- Üst sınır anahtarı (üsa)
- Alt sınır anahtarı (asa)

Gerçekleştirilen kartın çıkışları da aşağıda görülmektedir.

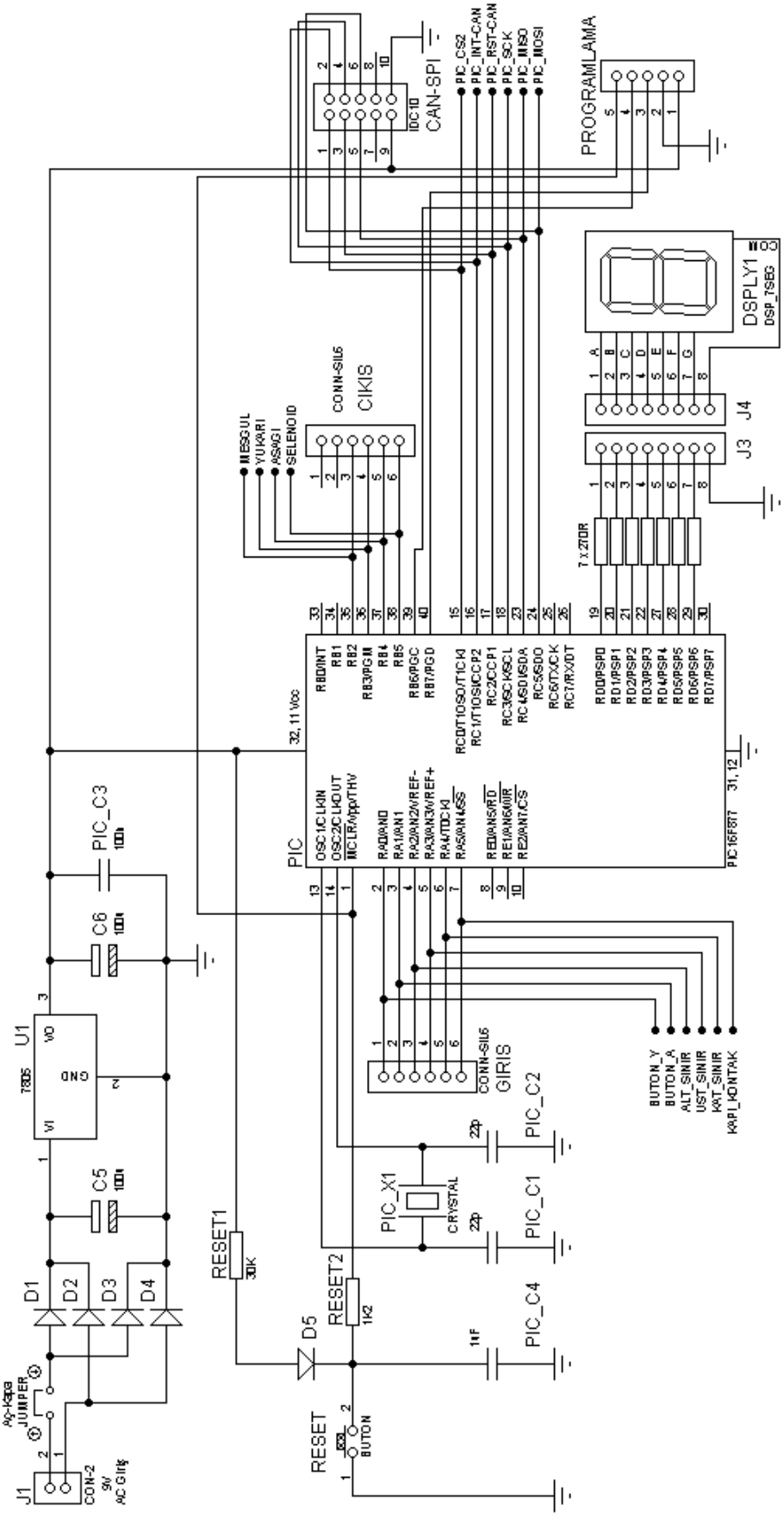
- Meşgul ledi
- Yukarı ledi
- Aşağı ledi
- Selenoid valf
- Yedi parçalı gösterge (7 segment display)

Çıkış olarak ayrıca programlama uçları alınmıştır. Yapılan çalışma içerisinde kartta güç ünitesinin olması kararlaştırılmış ve oluşturulan kartın devre şeması Şekil 6.2'de görülmektedir.

Bu tasarımda yedi parçalı gösterge kartın dışında gerçekleştirilmiştir. Bu sayede yedi parçalı gösterge, asansörün dış görünümüne taşınabilmektedir. Kat kontrol kartına ait baskılı devre ve yerleşim planı EK-F' de verilmiştir.

6.1.2 PIC16F877'li kabin kontrol kartı tasarımı

Kabin içi incelendiğinde, kabin içerisinde kat butonları ve imdat butonu bulunmaktadır. Ayrıca kabin içi çağrılan katın aydınlatmasının yapılması gerekmektedir. Bu bilgiler doğrultusunda kartın giriş ve çıkışları aşağıdaki gibi şekillenmiştir.



Şekil 6.2 Kat kontrol kartının açık şeması

Gerçekleştirilen kartın girişleri aşağıdaki görülmektedir.

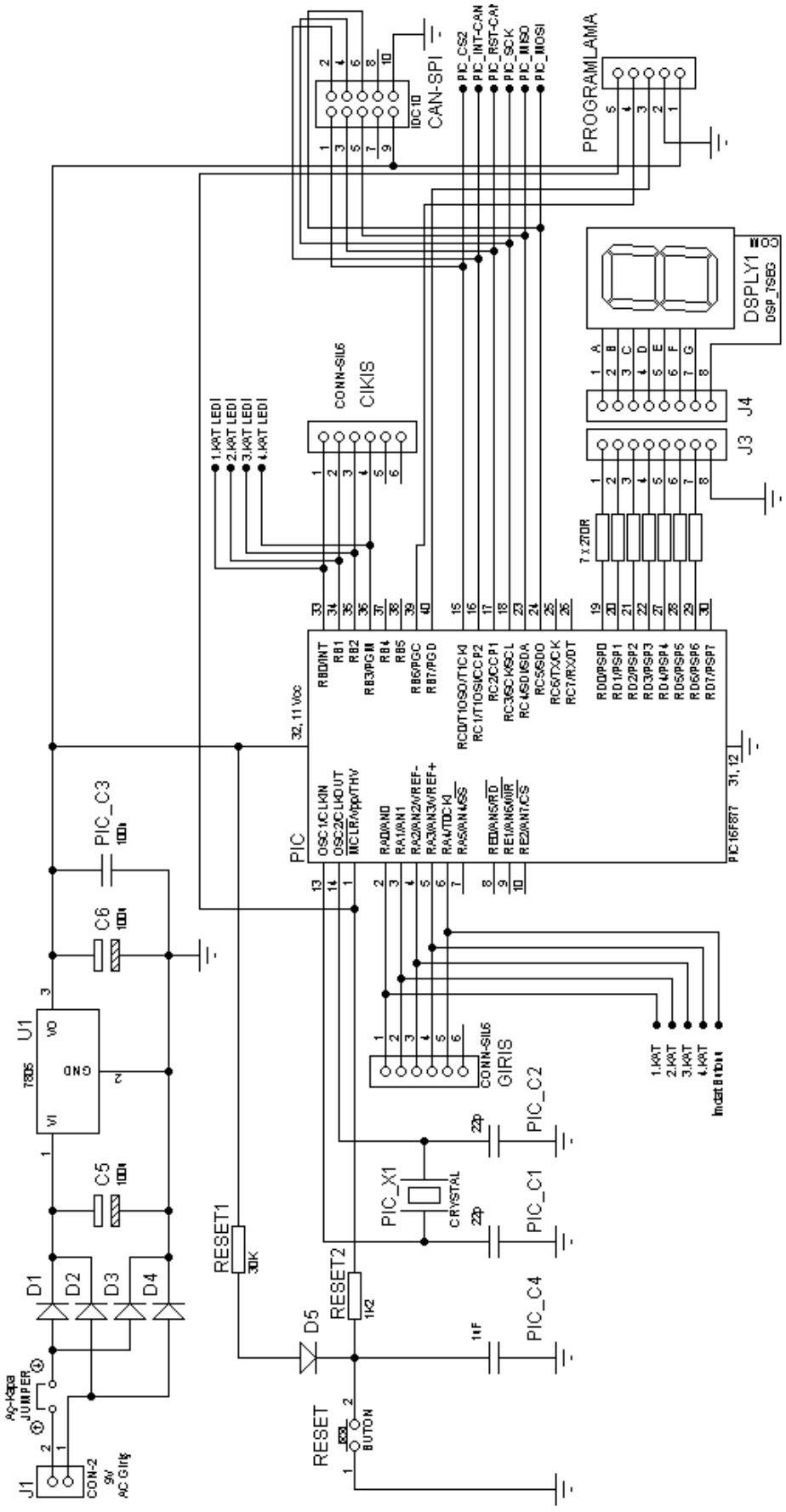
- Kat butonu (k1)
- Kat butonu (k2)
- Kat butonu (k3)
- Kat butonu (k4)
- İmdat butonu (ib)

Gerçekleştirilen kartın çıkışları aşağıdaki görülmektedir.

- 1. Kat ledi
- 2. Kat ledi
- 3. Kat ledi
- 4. Kat ledi
- Yedi parçalı gösterge (7 segment display)

PIC16F877 mikrodenetleyicisinin devre içerisinde programlanabilmesini sağlamak üzere programlama uçları için bağlantılar yapılmıştır. Yapılan çalışma da kartta güç ünitesinin bulunması kararlaştırılmıştır. Gerçekleştirilen kabin kontrol kartının açık şeması Şekil 6.3'te görülmektedir.

Bu tasarımda yedi parçalı gösterge kartın dışında gerçekleştirilmiştir. Bu sayede yedi parçalı gösterge, kabinin dış görünümüne taşınabilmektedir. Kabin kontrol kartına ait baskılı devre ve yerleşim planı EK-G'de verilmiştir.



Şekil 6.3 Kabin kontrol kartının açık şeması

6.1.3 PIC16F877'li ana kontrol kartı tasarımı

Ana kontrol ünitesinin içerisinde kullanılan kontrol bölümünün motoru kontrol etmesi gerekmektedir. Dört katlı model asansörün motorunun kontrol eden kısmı incelendiğinde, motorun aşağı yönde hareket için bir girişinin, yukarı yönde hareket için ayrı bir girişinin bulunduğu görülmüştür. Ayrıca kontrol ünitesinde oluşabilecek hataları düzenlemesinden sonra sistemin yeniden yüklenmesi için bir girişin olması kararlaştırılmıştır. Böylelikle hatalı bir durum olduğunda sistemi durdurabilecek ve hata düzeltildiğinde tekrar çalışması için komut verilebilecektir. Merkezi birim içerisinde de kat bilgilerini göstermek amacıyla yedi parçalı bir gösterge olması kararlaştırılmıştır. Bu doğrultuda merkezi birim için giriş ve çıkışlar aşağıdaki gibi olduğu tespit edilmiştir.

Gerçekleştirilen kartın girişi aşağıda görülmektedir.

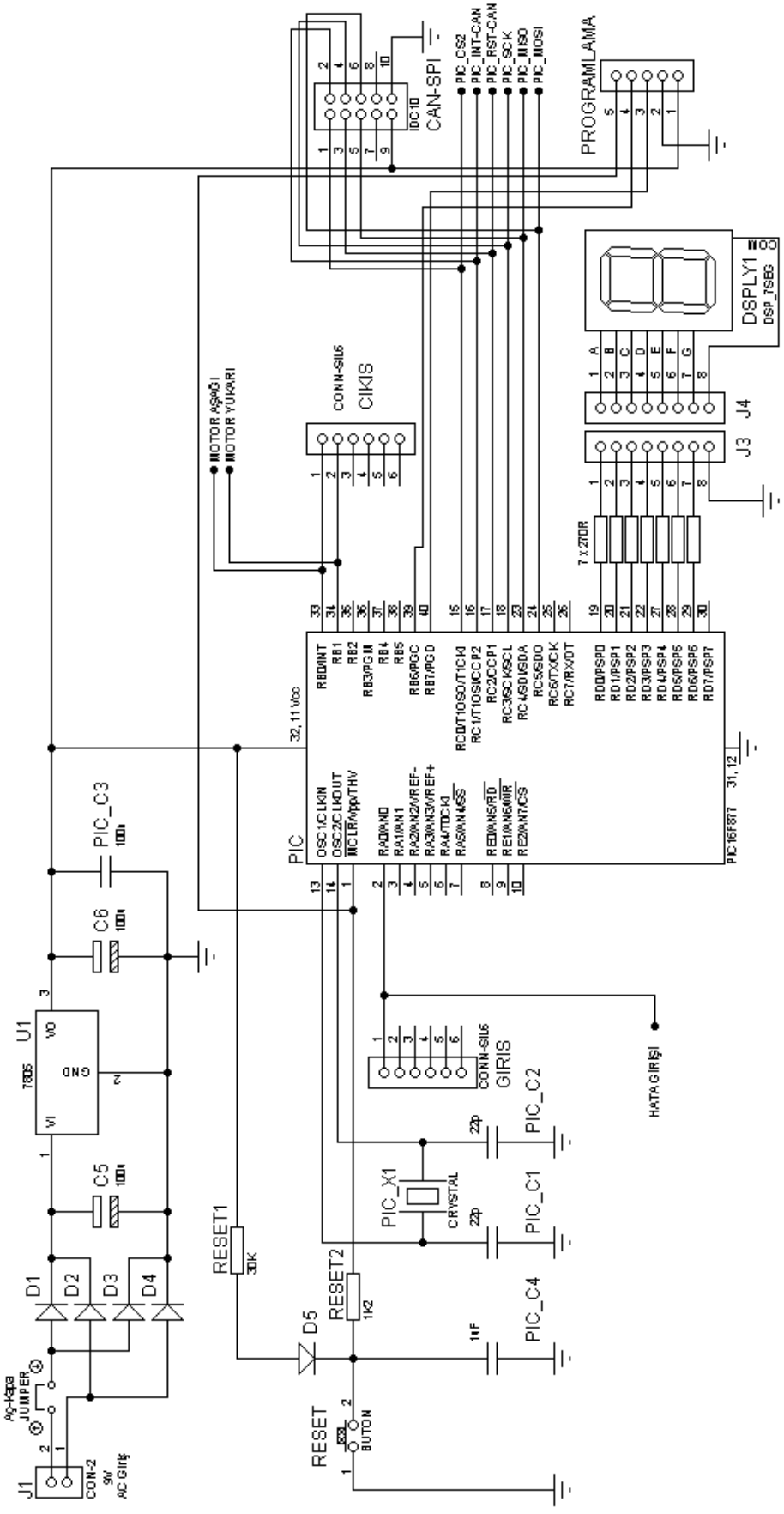
- Hata düzeltme girişi

Gerçekleştirilen kartın çıkışları da aşağıda görülmektedir.

- Motor aşağı
- Motor yukarı
- Yedi parçalı gösterge (7 segment display)

PIC16F877 mikrodenetleyicisinin devre içerisinde programlanabilmesini sağlamak üzere programlama uçları için bağlantılar yapılmıştır. Yapılan çalışmada kartta güç ünitesinin bulunması kararlaştırılmıştır. Gerçekleştirilen kabin kontrol kartının açık şeması Şekil 6.4'te görülmektedir.

Bu tasarımda yedi parçalı gösterge kartın dışında gerçekleştirilmiştir. Ana kontrol kartına ait baskılı devre ve yerleşim planı EK-H'de verilmiştir.

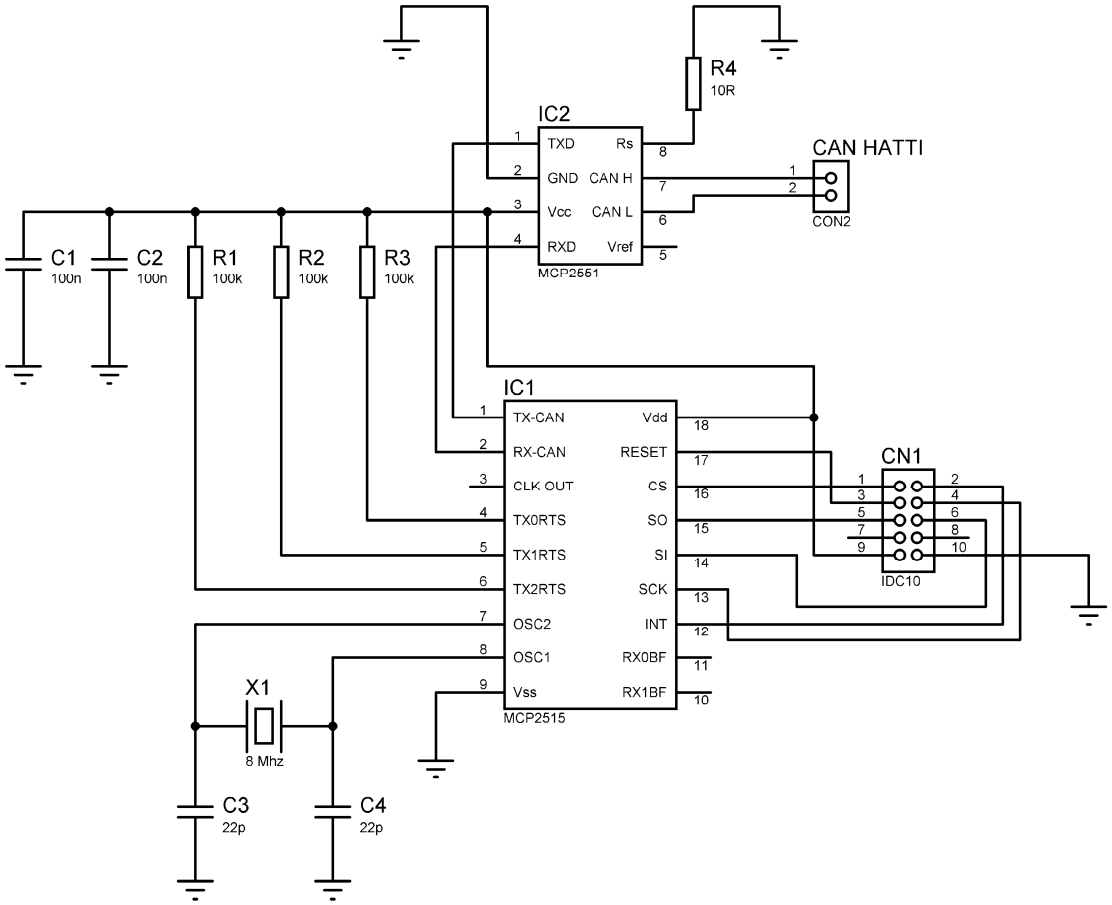


Şekil 6.4 Ana kontrol kartının açık şeması

6.1.4 CAN modülü kartı tasarımı

Dağıtık kontrol sisteminin bir parçaşı olan iletişim birimi olarak tercih edilen CAN-Bus haberleşme sistemi modülünde, Bölüm 4'te incelenen Microchip firmasının ürettiği MCP2515 ve MCP2551 entegreleri kullanılmıştır.

Gerçekleştirilen CAN modülü kartı açık şeması Şekil 6.5'te görülmektedir. CAN modülü kartına ait baskılı devre ve yerleşim planı EK-I'da verilmiştir.



Şekil 6.5 CAN modülü açık şeması

6.2 MicroBasic'te CAN komutları

Bu çalışmada yazılım geliştirmek için MicroBasic programlama dili kullanılmıştır. Geliştirilen yazılımın anlaşılmasına ışık tutmak için bu kısımda MicroBasic programlama dili içerisinde yer alan özel CAN komutları incelenmektedir.

6.2.1 CAN istek modunun ayarlanması

CAN istek modunun ayarlanması için CANSPISetOperationMode komutu kullanılmaktadır. Bu komut ile normal, uyku, döngü, dinleme ve ayarlama modları tespit edilebilir [16]. Bu komutun yapısı aşağıda verilmiştir.

```
Sub Procedue CANSPISetOperationMode (dim mode, wait_flag as byte)
```

Bu komut CAN'i istek moduna ayarlar. "mode" parametresi CAN_OP_MODE sabitlerinden biri olmalıdır. CAN_OP_MODE sabitleri CAN işlem durumlarını tanımlar. Bu sabitler aşağıda verilmiştir.

- Const CAN_MODE_BITS = $E0
- Const CAN_MODE_NORMAL = 0
- Const CAN_MODE_SLEEP = $20
- Const CAN_MODE_LOOP = $40
- Const CAN_MODE_LISTEN = $60
- Const CAN_MODE_CONFIG = $80

"wait_flag" parametresi 0 veya 0xFF olmalıdır. Eğer 0xFF'e ayarlı ise, bu bir engelli çağrıdır. İstek modu ayarlanmadığı sürece fonksiyon geri dönmeyecektir. Eğer 0'a ayarlı ise bu bir engelsiz çağrıdır. CAN biriminin istek moduna ayarlayıp ayarlanmadığına bakmayacaktır. Bu nedenle çağırıcı moda özgün işlemler yapmadan önce doğru çalışma modunun seçildiğini belirlemelidir [16].

Bu komutun kullanımını gösteren örnek kod aşağıda verilmiştir.

```
CANSPISetOperationMode (CAN_MODE_CONFIG, $FF)
```

6.2.2 CAN çalışma modunun tespiti

CAN biriminin geçerli çalışma modunun tespiti için CANSPIGetOperationMode komutu kullanılmaktadır [16]. Bu komutun yapısı aşağıda verilmiştir.

```
Sub function CANSPIGetOperationMode as byte
```

Bu komut ile işlem modu takip edilebilmektedir. Bu sayede moda özgün işlemler yapmadan önce bu komut ile mod tespiti gerçekleştirilir [16].

Bu komutun kullanımını gösteren örnek kod aşağıda verilmiştir.

```
if (CANSPIGetOperationMode = CAN_MODE_CONFIG) then …
```

6.2.3 CANSPI çalıştırma

CANSPI'yı başlatmak için CANSPIInitialize komutu kullanılır. Bu komut ile tüm bekleyen iletişimler iptal edilir. Tüm mesajların alınabilmesi için tüm maske kaydedicileri 0'a çekilir. CANSPI başlatılmadan önce SPI_Init (Bu komut ile özel SPI komutları çağırılır) komutu işlenmelidir. Ayrıca CANSPI, config durumunda olmalıdır. Aksi halde fonksiyon engellenecektir [16]. Bu komutun yapısı aşağıda görülmektedir.

```
Sub Procedure CANSPIInitialize (dim SJW, BRP, PHSEG1, PHSEG2, PROPSEG,
CAN_CONFIG_FLAGS as byte)
```

SJW, BRP, PHSEG1, PHSEG2 ve PROPSEG parametreleri, kullanılacak olan PIC'e özel değerlerdir ve doküman sayfalarında yer almaktadır. CAN_CONFIG_FLAGS parametresi ise sabitlerden oluşturulmaktadır [16].

Bu sabitler aşağıda verilmiştir.

- Const CAN_CONFIG_DEFAULT = $FF

- Const CAN_CONFIG_PHSEG2_PRG_BIT = $01
- Const CAN_CONFIG_PHSEG2_PRG_ON = $FF
- Const CAN_CONFIG_PHSEG2_PRG_OFF = $FE

- Const CAN_CONFIG_LINE_FILTER_BIT = $02
- Const CAN_CONFIG_LINE_FILTER_ON = $FF
- Const CAN_CONFIG_LINE_FILTER_OFF = $FD

- Const CAN_CONFIG_SAMPLE_BIT = $04
- Const CAN_CONFIG_SAMPLE_ONCE = $FF
- Const CAN_CONFIG_SAMPLE_THRICE = $FB
- Const CAN_CONFIG_MSG_TYPE_BIT = $08
- Const CAN_CONFIG_STD_MSG = $FF
- Const CAN_CONFIG_XTD_MSG = $F7

- Const CAN_CONFIG_DBL_BUFFER_BIT = $10
- Const CAN_CONFIG_DBL_BUFFER_ON = $FF
- Const CAN_CONFIG_DBL_BUFFER_OFF = $EF

- Const CAN_CONFIG_MSG_BITS = $60
- Const CAN_CONFIG_ALL_MSG = $FF
- Const CAN_CONFIG_VALID_XTD_MSG = $DF
- Const CAN_CONFIG_VALID_STD_MSG = $BF
- Const CAN_CONFIG_ALL_VALID_MSG = $9F

Bu komutun kullanımını gösteren örnek kod aşağıda verilmiştir.

```
init =  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_XTD_MSG      and
        CAN_CONFIG_LINE_FILTER_OFF

…

SPI_Init

…

CANSPIInitilialize (1, 1, 3, 3, 1, init)
```

6.2.4 CAN mesajlarını filtreleme

CAN hattından gelen mesajların filtrelenmesi için maskelerin ayarlanması gerekir. CANSPISetMask fonksiyonu, mesajları yüksek seviyede filtrelemek için maskeleri ayarlar [16]. Verilen değer tampon maske kaydedicisinin uygun bitini ayarlar. Bu komutun yapısı aşağıda verilmiştir.

```
Sub procedure CANSPISetMask (dim CAN_MASK as byte, dim value as longint, dim
CAN_CONFIG_FLAGS as byte)
```

Value parametresi maske kaydedicisinin değeridir. CAN_CONFIG_FLAGS parametresiyle filtrelenecek mesaj tipini seçer. Bu tip CAN_SONFIG_STD_MSG ya da CAN_SONFIG_XTD_MSG'den biridir. CAN_MASK parametresi önceden tanımlanmış sabitlerdendir [16]. Bu sabitler aşağıda görülmektedir.

- Const CAN_MASK_B1 = 0
- Const CAN_MASK_B2 = 1

Maskelendikten sonra veriler filtre edilebilir. Verileri filtre edebilmek için CANSPISetFilter komutu kullanılır. Bu fonksiyon, mesajları yüksek seviyede filtreler. Bu fonksiyonun yapısı aşağıda görülmektedir.

> Sub procedure CANSPISetFilter (dim CAN_FILTER as byte, dim value as longint, dim CAN_CONFIG_FLAGS as byte)

Burada CAN_CONFIG_FLAGS filtrelenecek mesaj tipini seçer. Value algılanılacak olan mesaj tanımlayıcı değeridir. CAN_FILTER parametresi önceden tanımlanan bir sabittir [16]. Bu sabitin değeri aşağıda verilmiştir.

- Const CAN_FILTER_B1_F1 = 0
- Const CAN_FILTER_B1_F2 = 1
- Const CAN_FILTER_B2_F1 = 2
- Const CAN_FILTER_B2_F2 = 3
- Const CAN_FILTER_B2_F3 = 4
- Const CAN_FILTER_B2_F4 = 5

Bu komutların kullanımını gösteren örnek kodlama aşağıda görülmektedir.

```
CANSPISetMask (CAN_MASK_B1,-1 , CAN_CONFIG_XTD_MSG)
'Tüm maske bitlerinin birler. Yani tüm filtreli bitler birlenir.

CANSPISetFilter (CAN_FILTER_B1_F1, 3 , CAN_CONFIG_XTD_MSG)
'B1_F1 filtresinin mesaj tanımlayıcısını 3' e kurar.
```

6.2.5 CAN mesajı gönderme

CAN mesajlarını göndermek için CANSPIWrite fonksiyonu kullanılmaktadır. Eğer en az bir boş gönderici tamponu bulunursa fonksiyon kuyruk üzerinden veri gönderimi için mesaj yollar. Eğer tampon dolu ise, fonksiyon sıfır üretir. Veri gönderimi gerçekleşene kadar bu sistemi döndürür [16]. CANSPI normal modda olmalıdır. Aksi halde mesaj iletimi olmayacaktır. Bu fonksiyonun yapısı aşağıda verilmiştir.

> Sub fonction CANSPIWrite (dim ID as longint, dim byref DATA as byte[8], dim DATAlen, CAN_TX_MSG_FLAGS as byte) as byte

ID parametresi CANSPI mesaj tanımlayıcısıdır. Mesaj tipine bağlı olarak sadece 11 veya 29 bit kullanılabilir. DATA parametresi 8 byte'a kadar uzayabilen byte'lar

dizisidir. Bu gönderilecek olan veridir. DATAlen parametresi ise 1-8 byte arasında veri uzunluğudur. Gönderilecek olan verinin uzunluğunu tespit etmektedir. CAN_TX_MSG_FLAGS sabitlerden düzenlenmiş değerdir. Burada kullanılabilen sabitler aşağıda verilmiştir.

- Const CAN_TX_PRIORITY_BITS = $03
- Const CAN_TX_PRIORITY_0 = $FC
- Const CAN_TX_PRIORITY_1 = $FD
- Const CAN_TX_PRIORITY_2 = $FE
- Const CAN_TX_PRIORITY_3 = $FF

- Const CAN_TX_FRAME_BIT = $08
- Const CAN_TX_STD_FRAME = $FF
- Const CAN_TX_XTD_FRAME = $F7

- Const CAN_TX_RTR_BIT = $40
- Const CAN_TX_NO_RTR_FRAME = $FF
- Const CAN_TX_RTR_FRAME = $BF

Bu komutun kullanımını gösteren örnek kod aşağıda verilmiştir.

```
tx = CAN_TX_PRIORITY_0 and CAN_TX_XTD_FRAME
CANSPIWrite (id, data, 2, tx)
```

6.2.6 CAN mesajı okuma

CAN mesajlarını okuyabilmek için CANSPIRead fonksiyonu kullanılmaktadır. Fonksiyon alıcı tamponundan mesaj okur. Eğer sonunda en az bir tam alıcı tampon bulunursa, bu ayrıştırılır ve işlenir. Eğer hiçbir tampon bulunmazsa, fonksiyon sıfır üretir. Bu fonksiyonun yapısı aşağıda verilmiştir. Bu fonksiyon için veri alımının uygun olduğu modda çalışma seçilmiş olmalıdır. Aksi halde mesaj okuması gerçekleşmeyecektir [16]. Bu fonksiyonun yapısı aşağıda verilmiştir.

```
Sub function CANSPIRead (dim byref ID as longint, dm byref DATA as byte[8], dim byref
DATAlen, CAN_RX_MSG_FLAGS as byte) as byte
```

ID mesaj tanımlayısıdır. DATA 8 byte'a kadar uzayabilen byte'lar dizisidir. DATAlen 1-8 byte arasında veri uzunluğudur. CAN_RX_MSG_FLAGS sabitlerden oluşmuş değerdir [16]. Bu sabitler aşağıda verilmiştir.

- Const CAN_RX_FILTER_BITS = $07 (Filtre bitlerine erişim için kullanılır.)
- Const CAN_RX_FILTER_1 = $00
- Const CAN_RX_FILTER_2 = $01
- Const CAN_RX_FILTER_3 = $02
- Const CAN_RX_FILTER_4 = $03
- Const CAN_RX_FILTER_5 = $04
- Const CAN_RX_FILTER_6 = $05
- Const CAN_RX_OWERFLOW_1 = $08 (Taşma varsa bir, aksi halde sıfır olur.)
- Const CAN_RX_INVALID_MSG = $10 (Geçersizse bir, yoksa sıfır olur.)
- Const CAN_RX_XTD_FRAME = $20 (XTD mesaj ise bir, yoksa sıfır olur.)
- Const CAN_RX_RTR_FRAME = $40 (RTR mesaj ise bir, yoksa sıfır olur.)
- Const CAN_RX_DBL_BUFFERED = $80 (Eğer donanımsal olarak çift tampon bellekliyse bir yap.)

Bu komutun kullanımını gösteren örnek kod aşağıda verilmiştir.

```
rcv = CANSPIRead (id, data, len, rx)
```

6.3 Programın Oluşturulması

Ana kontrol ünitesinin, kabin ünitesinin ve kat ünitelerinin yapması gereken işlemler dikkate alındığında, birbirinden farklı yapılara sahip üç program yapısı gerçekleştirilmiştir. Bu program yapıları bu kısımda anlatılmaktadır.

6.3.1 Katların program yapısı

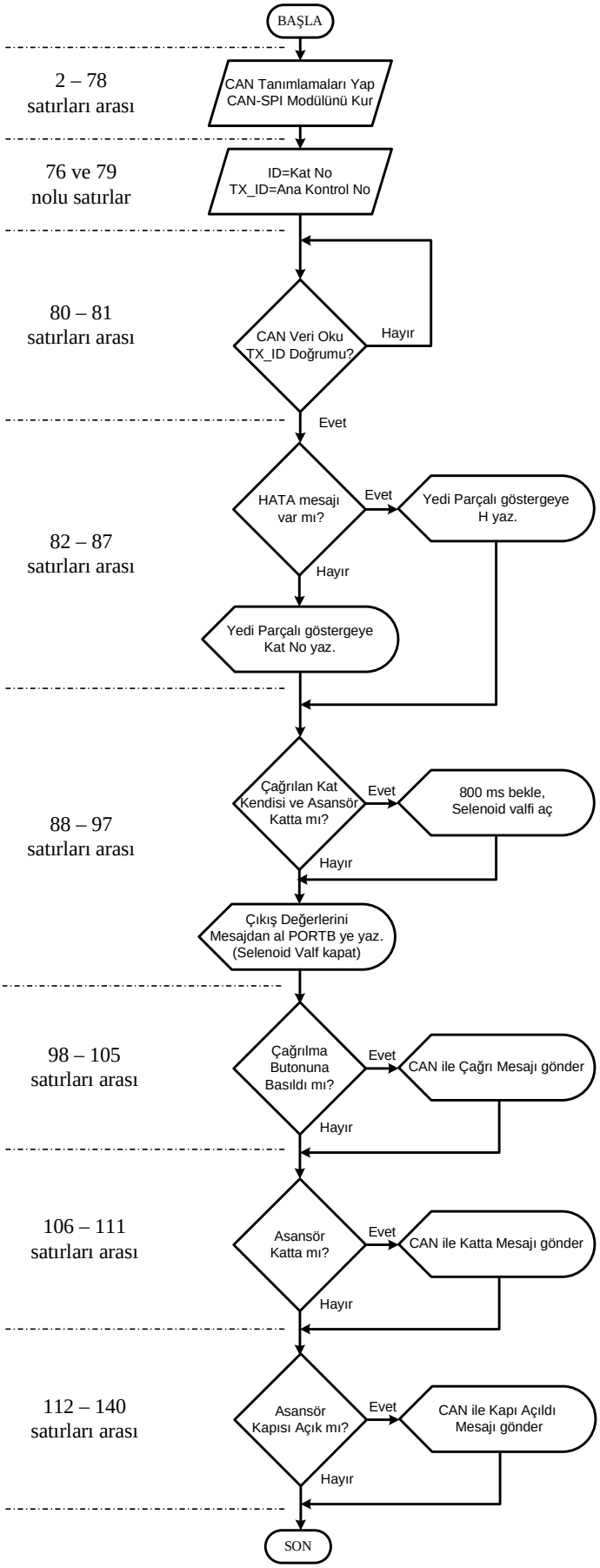
Katların kontrolünü sağlayacak olan ünitenin, gerçekleştirmesi istenilen görevler aşağıda görülmektedir.

- Kattaki kapının açık-kapalı bilgisini ana kontrol ünitesine CAN-Bus ile bildirmek,
- Kattan çağrıldı bilgisini ana kontrol ünitesine CAN-Bus ile göndermek,
- Asansörün bulunduğu katı yedi parçalı göstergeyle belirtmek,
- Asansör hareket halindeyken meşgul ledini, aşağı hareketi için aşağı ledini ve yukarı hareketi için yukarı ledini yakmak,
- Asansör çağrıldığı kata geldiğinde, ana kontrol ünitesine CAN-Bus ile kendi katının bilgisini göndermek,

- Asansör kendi katından çağırılıp, kendi katına geldiğinde selenoid valfini çalıştırmaktır.

Bu görevler incelendiğinde, kattaki ünitenin devamlı olarak CAN-Bus aracılığıyla ana üniteden gelen kat, çağrılma ve çıkış bilgilerini denetlemesi gerekmektedir. Bu bilgilerin devamında ise kendi katındaki giriş değerlerine uygun verileri ana kontrol ünitesine CAN-Bus aracılığı ile iletmelidir. Bu görevler incelenmiş ve çalışması için Şekil 6.6'daki çalışma algoritması oluşturulmuştur.

Şekil 6.6'daki algoritmaya uygun olarak yazılan MikroBasic program kodları birinci, ikinci, üçüncü ve dördüncü kat kontrol kartları için sırasıyla EK-C, EK-Ç, EK-D ve EK-H' de yer almaktadır.



Şekil 6.6 Kat kartlarının program algoritması ve program satır numaraları

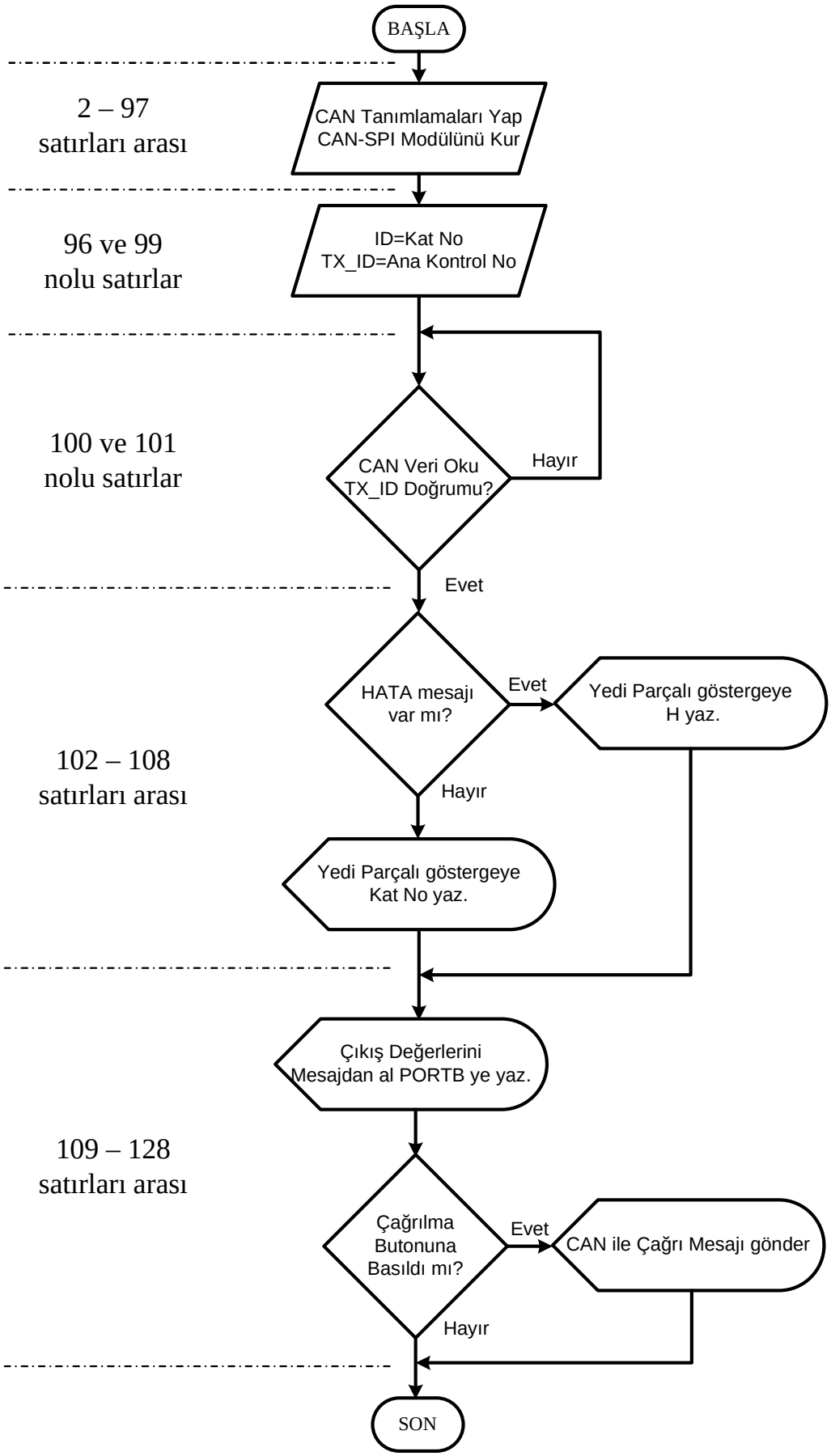
6.3.2 Kabin program yapısı

Kabin kontrolünü sağlayacak olan ünitenin, gerçekleştirmesi istenilen görevler aşağıda görülmektedir.

- Hangi kat butonuna basılırsa, o kat için çağrıldı bilgisini ana kontrol ünitesine CAN-Bus ile göndermek,
- Asansörün bulunduğu katı yedi parçalı göstergeyle belirtmek,
- Asansörün gittiği katın ledini yakmaktır.

Bu görevler incelendiğinde, kabin ünitesinin devamlı olarak CAN-Bus aracılığıyla ana üniteden gelen kat, çağrılma ve çıkış bilgilerini denetlemesi gerekmektedir. Bu bilgilerin devamında ise kendi katındaki giriş değerlerine uygun verileri ana kontrol ünitesine CAN-Bus aracılığı ile iletmelidir. Bu görevler incelenmiş ve çalışması için Şekil 6.7'deki çalışma algoritması oluşturulmuştur.

Şekil 6.7'deki algoritmaya uygun olarak kabin kontrol kartı için yazılan MikroBasic program kodları EK-B'de yer almaktadır.



Şekil 6.7 Kabin kartının program algoritması ve program satır numaraları

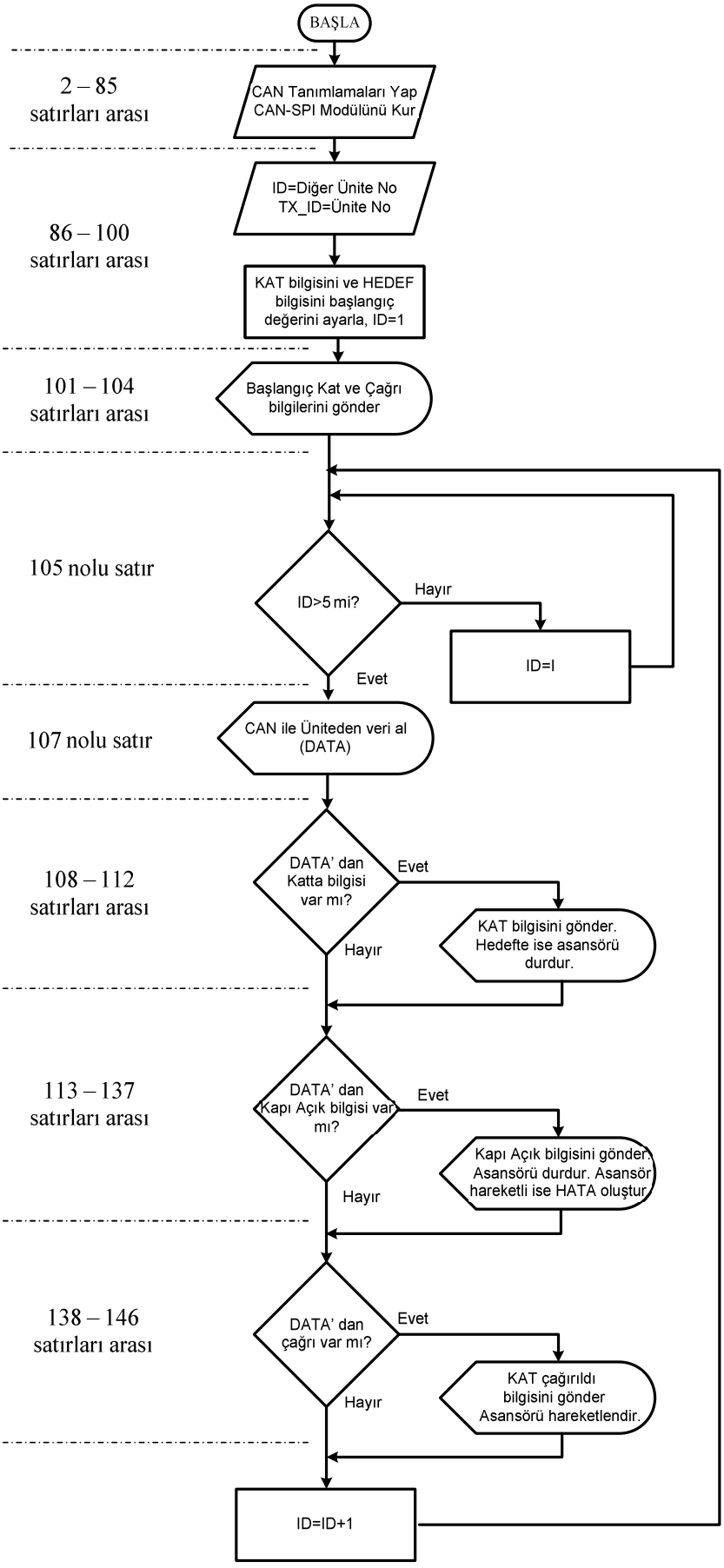
6.3.3 Ana kontrol program yapısı

Merkezi kontrolü sağlayacak olan bu ünitenin, gerçekleştirmesi istenilen görevler aşağıda görülmektedir.

- Motorun yukarı ve aşağı hareketini sağlamak,
- Katlardan CAN-Bus ile gelen çağrı mesajlarını değerlendirmek,
- Asansörün bulunduğu katı yedi parçalı göstergeyle belirtmek ve bu bilgiyi diğer kartlara CAN-Bus ile göndermek,
- Asansörün çağrıldığı kat bilgisini CAN-Bus ile diğer kartlara göndermek,
- Katlardan CAN-Bus ile gelen kapı açık bilgisini değerlendirmek,

Bu görevler incelendiğinde, ana ünitenin devamlı olarak CAN-Bus aracılığıyla diğer ünitelere kat bilgilerinin göndermesi gerekmektedir. Ayrıca kabinden ve katlardan gelen verileri değerlendirip diğer ünitelerin çalışmasını yönlendirmelidir. Bu görevler incelenmiş ve çalışması için Şekil 6.8'deki çalışma algoritması oluşturulmuştur.

Şekil 6.8'deki algoritmaya uygun olarak ana kontrol kartı için yazılan MikroBasic program kodları EK-A'da yer almaktadır.



Şekil 6.8 Ana kontrol kartının program algoritması ve program satır numaraları

BÖLÜM VII

SONUÇLAR

Bu tez çalışmasının amacı, CAN-Bus teknolojisini kullanarak bir dağıtık kontrol uygulamasının tasarlanması ve gerçekleştirilmesidir.

Dağıtık kontrol uygulaması olarak dört katlı model bir asansör kullanılmıştır. Her katta kapı güvenliğini sağlamak amacıyla kullanılan selenoid valf, bir kat sınır anahtarı, bir kapı anahtarı ve meşgul lambası bulunmaktadır. Ayrıca her katta kabin aşağı yönde gidiyor ya da kabin yukarı yönde gidiyor lambaları (LED'leri) kullanılmıştır. Kabin içerisinde kat butonları ve imdat butonu bulunmaktadır. Model asansörün üstündeki ayrı bir bölümde ise kabinin aşağı veya yukarı yönde hareketini sağlayan bir 12 V'luk DC motor mevcuttur.

Model asansörün dağıtık olarak kontrolünü gerçekleştirmek için; dört kat için birer kontrol kartı, bir adet kabin içi kontrol kartı ve bir adet de ana kontrol kartı gerçekleştirilmiştir. Her bir kontrol kartında bir PIC16F877 mikrodenetleyicisi kontrol ve haberleşmeyi sağlamak üzere kullanılmıştır. Kontrol kartlarının diğer kartlarla haberleşmesini CAN-Bus üzerinden sağlamak için her kontrol kartına ayrı ayrı bağlanmak üzere toplamda altı adet CAN modülü gerçekleştirilmiştir. Her bir CAN modülünde bir tane MCP2515 CAN-SPI ve bir tane MCP2551 CAN-Bus çevirici kullanılmıştır. Böylelikle devrede bulunan altı modülün CAN-Bus üzerinden haberleşerek dağıtık kontrol donanımı oluşturulmuştur. Sistemin kontrolünü sağlamak üzere gerekli yazılımlar MicroBasic programlama dili kullanılarak gerçekleştirilmiştir. Bu dilin seçilmesinin en önemli sebebi bu dilin kullanımının kolay olması ve özel CAN komutları içermesidir. Sonuçta model asansörün CAN-Bus teknolojisi kullanılarak dağıtık kontrolü başarılı bir şekilde gerçekleştirilmiştir. Böylelikle CAN-Bus teknolojisinin dağıtık kontrol uygulamaları için kullanışlı bir teknoloji olduğu gözlemlenmiştir.

KAYNAKLAR

[1] Sezgin ERSOY, Sezai TASKIN; "Otomotivde Can-Bus Teknolojisi ve Otomotiv Mekatroniğinin Geleceği"; Endüstri & Otomasyon Aylik Elektrik, Elektronik, Makina, Bilgisayar, Kontrol, Sistemleri Dergisi; Sayi: 117; ISSN: 1301 – 3610, Aralik 2006.

[2] Microchip Technology Inc., www.microchip.com

[3] Programming and Customizing PIC Microcontrollers, 2 nd Edition, PREDKO MYKE, 1161 pages, 2000.

[4] MCP250XX, Development Kit, User's Guide, Microchip Technology Inc. DS51266C, 2004.

[5] Understanding the MCP250XX Devices, Microchip Technology Inc. DS00815A, 2001.

[6] MCP250XX, Programming specifications, Microchip Technology Inc. DS20072B, 2001.

[7] Altınbaşak, O., PicBasic ile Programlama, Altaş Yayınları, 2007.

[8] Ergül, T., RF Multi Kontrol, Yüksek Lisans Tezi Yıldız Teknik Üniversitesi Elektronik Haberleşme Mühendisliği, 2009.

[9] Türkay, Y. ve Göksu, A., Mikrodenetleyiciler ve MicroBasic Ders Notları, 2008.

[10] Lawrenz W., Worldwide Status of CAN – Present and Future, Proceedings of 2nd international CAN Conference, 1995.

[11] Özçelik, İ., Ertürk, İ. ve Ekiz, H., CAN – Eternet Uyumlu Köprü Tasarımı ve Uygulaması, 2000.

[12] Ünal, İ., CAN Üzerinden PIC Programlama, Süleyman Demirel Üniversitesi Elektronik Bilgisayar Eğitimi Ana Bilim Dalı, 2006.

[13] Aksebzeci, B. Hakan, Bir Asansör Kontrol Sisteminin Petri Netler Yardımıyla Modellenmesi ve XILINX XC2S200E FPGA'sı ile Gerçekleştirilmesi, Yüksek Lisans Tezi Niğde Üniversitesi Fen Bilimleri Enstitüsü, 2005.

[14] Çolak, İ., Bayıdır, R. ve Özbilen, A., Dağıtık Denetim Sistemlerine Yönelik Elektronik Tehditler, TÜBAV Bilim Dergisi, Cilt 1 Sayı 2 2008.

[15] Pazul, K., Controller Area Network (CAN) Basics, Microchip Technology Inc., 1999.

[16] Köksal, F. Zeynep ve Samancı, K., PIC Mikrodenetleyiciler İçin MicroBasic Derleyicisi, Beti Bilişim Teknolojileri, 2006.

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

[18] Lawrenz, W., CAN System Engineering From Theory to Practical Applications, Springer-Verlag, 1997.

[19] Hopkins, A., Controller Area Networks, Computer Science Seminar Spring, 2003.

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

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

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

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

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

[25] Microchip Inc., High Speed CAN Transceiver Datasheet, Microchip Technology Inc., 2003.

[26] Microchip Inc., PIC 18FXX8 Datasheet, Microchip Technology Inc., 2004

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

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

[29] GÖRGÜLÜ, Y.E., RTX51 ile Asansör Otomasyonu, Yüksek Lisans Tezi Süleyman Demiral Üniversitesi Fen Bilimleri Enstitüsü, 1-3, 2007

EKLER

EK-A

Ana Kontrol Kartı Programının MicroBasic Kodları

```
program ANA
dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
      Rx_Data_Len as byte
      RxTx_Data as byte[8]
      Msg_Rcvd as byte
      Tx_ID, Rx_ID, ID as longint
      KAT, Hedef, Hata as byte
dim   CanSpi_CS as sbit at RC0_bit
      CanSpi_CS_Direction as sbit at TRISC0_bit
      CanSpi_Rst as sbit at RC2_bit
      CanSpi_Rst_Direction as sbit at TRISC2_bit
Sub Procedure KAT_Yaz
Select Case KAT
      Case 1
            PORTD=$06
      Case 2
            PORTD=$5B
      Case 3
            PORTD=$4F
      Case 4
            PORTD=$66
      Case 5
            PORTD=$6D
      Case 0
            PORTD=$76
      Case 254
            PORTD=$76
      Case 255
            PORTD=$01
            Delay_MS(45)
            PORTD=$02
            Delay_MS(45)
            PORTD=$04
            Delay_MS(45)
            PORTD=$08
            Delay_MS(45)
            PORTD=$10
            Delay_MS(45)
            PORTD=$20
End Select
end Sub

Sub Procedure Veri_Gon
      RxTx_data[0]=KAT
      RxTx_data[1]=Hedef
      RxTx_data[2]=PortB
      CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
      Delay_ms(1)
End Sub
Sub Procedure Can_Cfg
      CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
```

```
52          CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_ALL_MSG)
53          CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_ALL_MSG)
54          CANSPISetFilter(_CANSPI_FILTER_B2_F4,ID,_CANSPI_CONFIG_ALL_MSG)
55          CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)
56    End Sub
57    main:
58    ADCON1=7
59    PORTB = 0
60    TRISB = 0
61    PORTA=0
62    TRISA=$FF
63    PORTD=0
64    TRISD=0
65
66    Can_Init_Flags = 0
67    Can_Send_Flags = 0
68    Can_Rcv_Flags = 0
69    Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
70                          _CANSPI_TX_XTD_FRAME and
71                          _CANSPI_TX_NO_RTR_FRAME
72
73    Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
74                          _CANSPI_CONFIG_PHSEG2_PRG_ON and
75                          _CANSPI_CONFIG_ALL_MSG and
76                          _CANSPI_CONFIG_DBL_BUFFER_ON and
77                          _CANSPI_CONFIG_ALL_MSG
78    Tx_ID = 12111
79    SPI1_Init()
80    CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
81    KAT=255
82    Hedef=255
83    PortB=$00
84    RxTx_data[3]=$00
85    Basla:
86    if ((PortA.0=1) and (Hata=$FF)) then
87          Hedef=4
88          KAT=254
89          RxTx_data[0]=$FF
90          RxTx_data[1]=$04
91          RxTx_data[2]=$00
92          RxTx_data[3]=$00
93    end if
94    if ((PortA.0=1) and (Hata=$F0)) then
95          Hedef=1
96          KAT=0
97          RxTx_data[0]=$00
98          RxTx_data[1]=$01
99          RxTx_data[2]=$00
100         RxTx_data[3]=$00
101   end if
102   if ((Hedef<KAT) and (RxTx_data[3]=$00)) then PORTB=$15 end if
103   if ((Hedef>KAT) and (RxTx_data[3]=$00)) then PORTB=$0E end if
104   if ((Hedef=KAT) and (RxTx_data[3]=$00)) then PORTB=$00 end if
105   for ID=1 to 5
106         Can_Cfg
107         Msg_Rcvd=CANSPIRead(Rx_ID, RxTx_Data, Rx_Data_Len, Can_Rcv_Flags)
108         if ((Rx_ID=ID) and Msg_Rcvd) then
109               if ((KAT=255) and (Hedef=255)) then
110                     KAT=RxTx_data[0]
111                     Hedef=KAT
```

```
112                 end if
113                 if (PortB=$00) then
114                       if (Hedef=RxTx_data[1]) then
115                             KAT=RxTx_data[0]
116                             Hedef=KAT
117                       else
118                             Hedef=RxTx_data[1]
119                       end if
120                 else
121                       KAT=RxTx_data[0]
122                 end if
123                 if RxTx_data[3]=$00 then
124                       if Hedef<KAT then PORTB=$15 end if
125                       if Hedef>KAT then PORTB=$0E end if
126                       if Hedef=KAT then PORTB=$00 end if
127                 end if
128                 if((RxTx_data[3]<>$F0)and(RxTx_data[3]<>$FF)and(RxTx_data[3]<>$00)) then
129                       KAT=RxTx_data[3]
130                       Kat_Yaz
131                       Delay_ms(300)
132                       KAT=255
133                       Kat_Yaz
134                       KAT=RxTx_data[0]
135                       Kat_Yaz
136                 end if
137           End if
138           if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
139                 Hata=RxTx_data[3]
140                 PORTB=$00
141                 PORTD=$76
142                 KAT=$FF
143                 Hedef=$FF
144           else
145                 Kat_Yaz
146           end if
147           Veri_Gon
148     Next ID
149     Goto Basla
150     end.
```

EK-B

Kabin Kontrol Kartı Programının MicroBasic Kodları

```
1    program KABIN
2
3    dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
4          Rx_Data_Len as byte
5          RxTx_Data as byte[8]
6          Msg_Rcvd as byte
7          Tx_ID, Rx_ID, KAT, Hedef as longint
8
9    dim   CanSpi_CS as sbit at RC0_bit
10         CanSpi_CS_Direction as sbit at TRISC0_bit
11         CanSpi_Rst as sbit at PORTC.B2
12         CanSpi_Rst_Direction as sbit at TRISC2_bit
13
14   Sub Procedure KAT_Yaz
15   Select Case KAT
16         Case 0
17               PORTD=$3F
18         Case 1
19               PORTD=$06
20         Case 2
21               PORTD=$5B
22         Case 3
23               PORTD=$4F
24         Case 4
25               PORTD=$66
26         Case 5
27               PORTD=$6D
28         Case else
29               PORTD=$01
30               Delay_MS(45)
31               PORTD=$02
32               Delay_MS(45)
33               PORTD=$04
34               Delay_MS(45)
35               PORTD=$08
36               Delay_MS(45)
37               PORTD=$10
38               Delay_MS(45)
39               PORTD=$20
40   End Select
41   end Sub
42
43   Sub Procedure HEDEF_Yaz
44   Select Case Hedef
45         Case 0
46               PORTB=$00
47         Case 1
48               PORTB=$01
49         Case 2
50               PORTB=$02
51         Case 3
52               PORTB=$04
53         Case 4
54               PORTB=$08
55         Case 5
```

```
56              PORTB=$10
57          Case Else
58              PORTB=$00
59      End Select
60      end Sub
61
62      Sub Procedure Veri_Gon
63          CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
64          Delay_ms(1)
65      end Sub
66
67      main:
68      ADCON1=7
69      PORTB = 0
70      TRISB = 0
71      PORTA=0
72      TRISA=$FF
73      PORTD=0
74      TRISD=0
75
76      Can_Init_Flags = 0
77      Can_Send_Flags = 0
78      Can_Rcv_Flags = 0
79
80      Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
81                            _CANSPI_TX_XTD_FRAME and
82                            _CANSPI_TX_NO_RTR_FRAME
83
84      Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
85                            _CANSPI_CONFIG_PHSEG2_PRG_ON and
86                            _CANSPI_CONFIG_XTD_MSG and
87                            _CANSPI_CONFIG_DBL_BUFFER_ON and
88                            _CANSPI_CONFIG_VALID_XTD_MSG and
89                            _CANSPI_CONFIG_LINE_FILTER_OFF
90
91      SPI1_Init()
92      CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
93      CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
94      CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)
95      CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)
96      CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG)
97      CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)
98
99      Tx_ID = 5
100     while TRUE
101         Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
102         if ((Rx_ID=12111) and Msg_Rcvd) then
103             KAT=RxTX_Data[0]
104             if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
105                 PORTD=$76
106             else
107                 Kat_Yaz
108             end if
110             Hedef=RxTX_Data[1]
111             if Hedef<>KAT then Hedef_Yaz else PORTB=0 end if
112         end if
113         if PortA.0=1 then
114             RxTX_Data[1]=$01
115             Veri_Gon
116         end if
```

```
117         if PortA.1=1 then
118                 RxTX_Data[1]=$02
119                 Veri_Gon
120         end if
121         if PortA.2=1 then
122                 RxTX_Data[1]=$03
123                 Veri_Gon
124         end if
125         if PortA.3=1 then
126                 RxTX_Data[1]=$04
127                 Veri_Gon
128         end if
129 wend
130 end.
```

EK-C

Birinci Kat Kontrol Kartı Programının MicroBasic Kodları

```
1   program KAT1
2
3   dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
4         Rx_Data_Len as byte
5         RxTx_Data as byte[8]
6         Msg_Rcvd as byte
7         Tx_ID, Rx_ID, KAT as longint
8
9   dim   CanSpi_CS as sbit at RC0_bit
10        CanSpi_CS_Direction as sbit at TRISC0_bit
11        CanSpi_Rst as sbit at PORTC.B2
12        CanSpi_Rst_Direction as sbit at TRISC2_bit
13
14  Sub Procedure KAT_Yaz
15  Select Case KAT
16        Case 0
17              PORTD=$3F
18        Case 1
19              PORTD=$06
20        Case 2
21              PORTD=$5B
22        Case 3
23              PORTD=$4F
24        Case 4
25              PORTD=$66
26        Case 5
27              PORTD=$6D
28        Case else
29              PORTD=$01
30              Delay_MS(45)
31              PORTD=$02
32              Delay_MS(45)
33              PORTD=$04
34              Delay_MS(45)
35              PORTD=$08
36              Delay_MS(45)
37              PORTD=$10
38              Delay_MS(45)
39              PORTD=$20
40  End Select
41  end Sub
42
43  Sub Procedure Veri_Gon
44        CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
45        Delay_ms(1)
46  end Sub
47
48  main:
49  ADCON1=7
50  PORTB = 0
51  TRISB = 0
52  PORTA=0
53  TRISA=$FF
54  PORTD=0
55  TRISD=0
```

```
56  Can_Init_Flags = 0
57  Can_Send_Flags = 0
58  Can_Rcv_Flags = 0
59
60  Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
61                        _CANSPI_TX_XTD_FRAME and
62                        _CANSPI_TX_NO_RTR_FRAME
63
64  Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
65                        _CANSPI_CONFIG_PHSEG2_PRG_ON and
66                        _CANSPI_CONFIG_XTD_MSG and
67                        _CANSPI_CONFIG_DBL_BUFFER_ON and
68                        _CANSPI_CONFIG_VALID_XTD_MSG and
69                        _CANSPI_CONFIG_LINE_FILTER_OFF
70
71  SPI1_Init()
72  CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
73  CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
74  CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)
75  CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)
76  CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG)
77  CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)
78
79  Tx_ID = 1
80  while TRUE
81      Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
82      if ((Rx_ID=12111) and Msg_Rcvd) then
83          KAT=RxTX_Data[0]
84          if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
85              PORTD=$76
86          else
87              Kat_Yaz
88          end if
89          if KAT=Tx_ID then
90              if RxTx_data[2]=$00 then
91                  if PortB.5=0 then
92                      PORTB=RxTX_Data[2]
93                      delay_ms(800)
94                      PORTB.5=1
95                  end if
96              else
97                  PORTB=RxTX_Data[2]
98              end if
99          else
100             PORTB=RxTX_Data[2]
101         end if
102     end if
103     if ((PortA.4=1) or (PortA.5=1)) then
104         RxTX_Data[1]=Tx_ID
105         Veri_Gon
106     end if
107     if ((PortA.0=1) and (PortB.5=0)and (KAT=255)) then
108         RxTX_data[0]=Tx_ID
109         RxTX_Data[1]=Tx_ID
110         Veri_Gon
111     end if
112     if ((PortA.0=1) and (PortB.5=0)and (RxTX_Data[2]<>$00)) then
113         RxTX_data[0]=Tx_ID
114         Veri_Gon
115     end if
```

```
116         if ((PortA.1=1) and (RxTx_data[3]=Tx_ID)) then
117             RxTx_data[3]=$00
118             Veri_Gon
119         end if
120         if PortA.1=0 then
121             RxTx_data[3]=Tx_ID
122             Veri_Gon
123         end if
124         if PortA.2=1 then
125             RxTx_data[3]=$FF
126             Veri_Gon
127         end if
128         if PortA.3=1 then
129             RxTx_data[3]=$F0
130             Veri_Gon
131         end if
132         if ((RxTx_data[3]<>$F0) and (RxTx_data[3]<>$FF) and (RxTx_data[3]<>$00)) then
133             Delay_ms(300)
134             KAT=RxTx_data[3]
135             Kat_Yaz
136             Delay_ms(300)
137             KAT=255
138             Kat_Yaz
139             KAT=RxTx_data[0]
140             Kat_Yaz
141         end if
142 wend
143 end.
```

EK-Ç

İkinci Kat Kontrol Kartı Programının MicroBasic Kodları

```
1    program KAT2
2
3    dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
4          Rx_Data_Len as byte
5          RxTx_Data as byte[8]
6          Msg_Rcvd as byte
7          Tx_ID, Rx_ID, KAT as longint
8
9    dim   CanSpi_CS as sbit at RC0_bit
10         CanSpi_CS_Direction as sbit at TRISC0_bit
11         CanSpi_Rst as sbit at PORTC.B2
12         CanSpi_Rst_Direction as sbit at TRISC2_bit
13
14   Sub Procedure KAT_Yaz
15   Select Case KAT
16         Case 0
17               PORTD=$3F
18         Case 1
19               PORTD=$06
20         Case 2
21               PORTD=$5B
22         Case 3
23               PORTD=$4F
24         Case 4
25               PORTD=$66
26         Case 5
27               PORTD=$6D
28         Case else
29               PORTD=$01
30               Delay_MS(45)
31               PORTD=$02
32               Delay_MS(45)
33               PORTD=$04
34               Delay_MS(45)
35               PORTD=$08
36               Delay_MS(45)
37               PORTD=$10
38               Delay_MS(45)
39               PORTD=$20
40   End Select
41   end Sub
42
43   Sub Procedure Veri_Gon
44         CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
45         Delay_ms(1)
46   end Sub
47
48   main:
49   ADCON1=7
50   PORTB = 0
51   TRISB = 0
52   PORTA=0
53   TRISA=$FF
54   PORTD=0
55   TRISD=0
```

```
Can_Init_Flags = 0
Can_Send_Flags = 0
Can_Rcv_Flags = 0

Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
                      _CANSPI_TX_XTD_FRAME and
                      _CANSPI_TX_NO_RTR_FRAME

Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
                      _CANSPI_CONFIG_PHSEG2_PRG_ON and
                      _CANSPI_CONFIG_XTD_MSG and
                      _CANSPI_CONFIG_DBL_BUFFER_ON and
                      _CANSPI_CONFIG_VALID_XTD_MSG and
                      _CANSPI_CONFIG_LINE_FILTER_OFF

SPI1_Init()
CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG)
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)

Tx_ID = 2
while TRUE
      Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
      if ((Rx_ID=12111) and Msg_Rcvd) then
            KAT=RxTX_Data[0]
            if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
                  PORTD=$76
            else
                  Kat_Yaz
            end if
            if KAT=Tx_ID then
                  if RxTx_data[2]=$00 then
                        if PortB.5=0 then
                              PORTB=RxTX_Data[2]
                              delay_ms(800)
                              PORTB.5=1
                        end if
                  else
                        PORTB=RxTX_Data[2]
                  end if
            else
                  PORTB=RxTX_Data[2]
            end if
      end if
      if ((PortA.4=1) or (PortA.5=1)) then
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (KAT=255)) then
            RxTX_data[0]=Tx_ID
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (RxTX_Data[2]<>$00)) then
            RxTX_data[0]=Tx_ID
            Veri_Gon
      end if
```

```
116         if ((PortA.1=1) and (RxTx_data[3]=Tx_ID)) then
117             RxTx_data[3]=$00
118             Veri_Gon
119         end if
120         if PortA.1=0 then
121             RxTx_data[3]=Tx_ID
122             Veri_Gon
123         end if
124         if PortA.2=1 then
125             RxTx_data[3]=$FF
126             Veri_Gon
127         end if
128         if PortA.3=1 then
129             RxTx_data[3]=$F0
130             Veri_Gon
131         end if
132         if ((RxTx_data[3]<>$F0) and (RxTx_data[3]<>$FF) and (RxTx_data[3]<>$00)) then
133             Delay_ms(300)
134             KAT=RxTx_data[3]
135             Kat_Yaz
136             Delay_ms(300)
137             KAT=255
138             Kat_Yaz
139             KAT=RxTx_data[0]
140             Kat_Yaz
141         end if
142 wend
143 end.
```

EK-D

Üçüncü Kat Kontrol Kartı Programının MicroBasic Kodları

```
1    program KAT3
2
3    dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
4          Rx_Data_Len as byte
5          RxTx_Data as byte[8]
6          Msg_Rcvd as byte
7          Tx_ID, Rx_ID, KAT as longint
8
9    dim   CanSpi_CS as sbit at RC0_bit
10         CanSpi_CS_Direction as sbit at TRISC0_bit
11         CanSpi_Rst as sbit at PORTC.B2
12         CanSpi_Rst_Direction as sbit at TRISC2_bit
13
14   Sub Procedure KAT_Yaz
15   Select Case KAT
16         Case 0
17               PORTD=$3F
18         Case 1
19               PORTD=$06
20         Case 2
21               PORTD=$5B
22         Case 3
23               PORTD=$4F
24         Case 4
25               PORTD=$66
26         Case 5
27               PORTD=$6D
28         Case else
29               PORTD=$01
30               Delay_MS(45)
31               PORTD=$02
32               Delay_MS(45)
33               PORTD=$04
34               Delay_MS(45)
35               PORTD=$08
36               Delay_MS(45)
37               PORTD=$10
38               Delay_MS(45)
39               PORTD=$20
40   End Select
41   end Sub
42
43   Sub Procedure Veri_Gon
44         CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
45         Delay_ms(1)
46   end Sub
47
48   main:
49   ADCON1=7
50   PORTB = 0
51   TRISB = 0
52   PORTA=0
53   TRISA=$FF
54   PORTD=0
55   TRISD=0
```

```
Can_Init_Flags = 0
Can_Send_Flags = 0
Can_Rcv_Flags = 0

Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
                      _CANSPI_TX_XTD_FRAME and
                      _CANSPI_TX_NO_RTR_FRAME

Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
                      _CANSPI_CONFIG_PHSEG2_PRG_ON and
                      _CANSPI_CONFIG_XTD_MSG and
                      _CANSPI_CONFIG_DBL_BUFFER_ON and
                      _CANSPI_CONFIG_VALID_XTD_MSG and
                      _CANSPI_CONFIG_LINE_FILTER_OFF

SPI1_Init()
CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG)
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)

Tx_ID = 3
while TRUE
      Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
      if ((Rx_ID=12111) and Msg_Rcvd) then
            KAT=RxTX_Data[0]
            if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
                  PORTD=$76
            else
                  Kat_Yaz
            end if
            if KAT=Tx_ID then
                  if RxTx_data[2]=$00 then
                        if PortB.5=0 then
                              PORTB=RxTX_Data[2]
                              delay_ms(800)
                              PORTB.5=1
                        end if
                  else
                        PORTB=RxTX_Data[2]
                  end if
            else
                  PORTB=RxTX_Data[2]
            end if
      end if
      if ((PortA.4=1) or (PortA.5=1)) then
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (KAT=255)) then
            RxTX_data[0]=Tx_ID
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (RxTX_Data[2]<>$00)) then
            RxTX_data[0]=Tx_ID
            Veri_Gon
      end if
```

```
116         if ((PortA.1=1) and (RxTx_data[3]=Tx_ID)) then
117             RxTx_data[3]=$00
118             Veri_Gon
119         end if
120         if PortA.1=0 then
121             RxTx_data[3]=Tx_ID
122             Veri_Gon
123         end if
124         if PortA.2=1 then
125             RxTx_data[3]=$FF
126             Veri_Gon
127         end if
128         if PortA.3=1 then
129             RxTx_data[3]=$F0
130             Veri_Gon
131         end if
132         if ((RxTx_data[3]<>$F0) and (RxTx_data[3]<>$FF) and (RxTx_data[3]<>$00)) then
133             Delay_ms(300)
134             KAT=RxTx_data[3]
135             Kat_Yaz
136             Delay_ms(300)
137             KAT=255
138             Kat_Yaz
139             KAT=RxTx_data[0]
140             Kat_Yaz
141         end if
142 wend
143 end.
```

EK-E

Dördüncü Kat Kontrol Kartı Programının MicroBasic Kodları

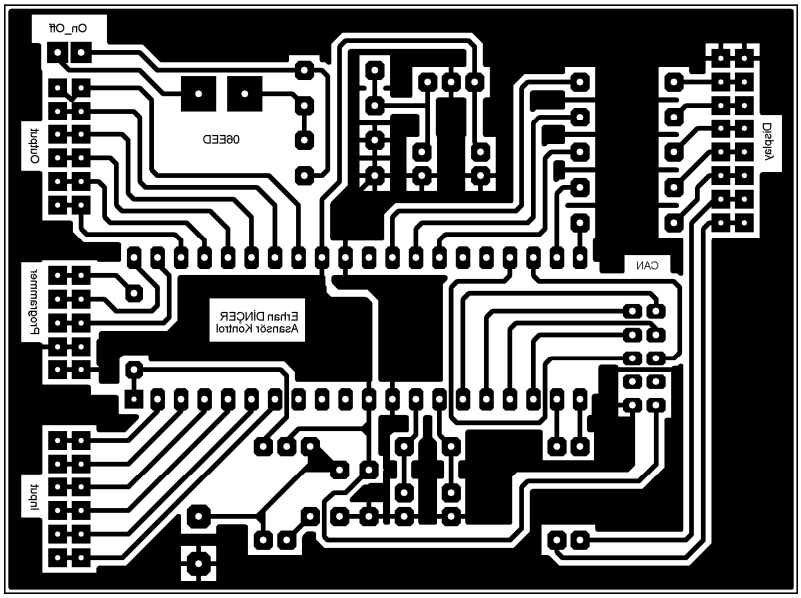
```
1    program KAT4
2
3    dim   Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
4          Rx_Data_Len as byte
5          RxTx_Data as byte[8]
6          Msg_Rcvd as byte
7          Tx_ID, Rx_ID, KAT as longint
8
9    dim   CanSpi_CS as sbit at RC0_bit
10         CanSpi_CS_Direction as sbit at TRISC0_bit
11         CanSpi_Rst as sbit at PORTC.B2
12         CanSpi_Rst_Direction as sbit at TRISC2_bit
13
14   Sub Procedure KAT_Yaz
15   Select Case KAT
16         Case 0
17               PORTD=$3F
18         Case 1
19               PORTD=$06
20         Case 2
21               PORTD=$5B
22         Case 3
23               PORTD=$4F
24         Case 4
25               PORTD=$66
26         Case 5
27               PORTD=$6D
28         Case else
29               PORTD=$01
30               Delay_MS(45)
31               PORTD=$02
32               Delay_MS(45)
33               PORTD=$04
34               Delay_MS(45)
35               PORTD=$08
36               Delay_MS(45)
37               PORTD=$10
38               Delay_MS(45)
39               PORTD=$20
40   End Select
41   end Sub
42
43   Sub Procedure Veri_Gon
44         CANSPIWrite(Tx_ID, RxTx_Data, 4, Can_Send_Flags)
45         Delay_ms(1)
46   end Sub
47
48   main:
49   ADCON1=7
50   PORTB = 0
51   TRISB = 0
52   PORTA=0
53   TRISA=$FF
54   PORTD=0
55   TRISD=0
```

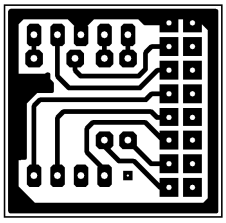
```
Can_Init_Flags = 0
Can_Send_Flags = 0
Can_Rcv_Flags = 0

Can_Send_Flags =      _CANSPI_TX_PRIORITY_0 and
                      _CANSPI_TX_XTD_FRAME and
                      _CANSPI_TX_NO_RTR_FRAME

Can_Init_Flags =      _CANSPI_CONFIG_SAMPLE_THRICE and
                      _CANSPI_CONFIG_PHSEG2_PRG_ON and
                      _CANSPI_CONFIG_XTD_MSG and
                      _CANSPI_CONFIG_DBL_BUFFER_ON and
                      _CANSPI_CONFIG_VALID_XTD_MSG and
                      _CANSPI_CONFIG_LINE_FILTER_OFF

SPI1_Init()
CANSPIInitialize(1,3,3,3,1,Can_Init_Flags)
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF)
CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG)
CANSPISetFilter(_CANSPI_FILTER_B2_F3,12111,_CANSPI_CONFIG_XTD_MSG)
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF)

Tx_ID = 4
while TRUE
      Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags)
      if ((Rx_ID=12111) and Msg_Rcvd) then
            KAT=RxTX_Data[0]
            if ((RxTx_data[3]=$F0) or (RxTx_data[3]=$FF)) then
                  PORTD=$76
            else
                  Kat_Yaz
            end if
            if KAT=Tx_ID then
                  if RxTx_data[2]=$00 then
                        if PortB.5=0 then
                              PORTB=RxTX_Data[2]
                              delay_ms(800)
                              PORTB.5=1
                        end if
                  else
                        PORTB=RxTX_Data[2]
                  end if
            else
                  PORTB=RxTX_Data[2]
            end if
      end if
      if ((PortA.4=1) or (PortA.5=1)) then
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (KAT=255)) then
            RxTX_data[0]=Tx_ID
            RxTX_Data[1]=Tx_ID
            Veri_Gon
      end if
      if ((PortA.0=1) and (PortB.5=0)and (RxTX_Data[2]<>$00)) then
            RxTX_data[0]=Tx_ID
            Veri_Gon
      end if
```

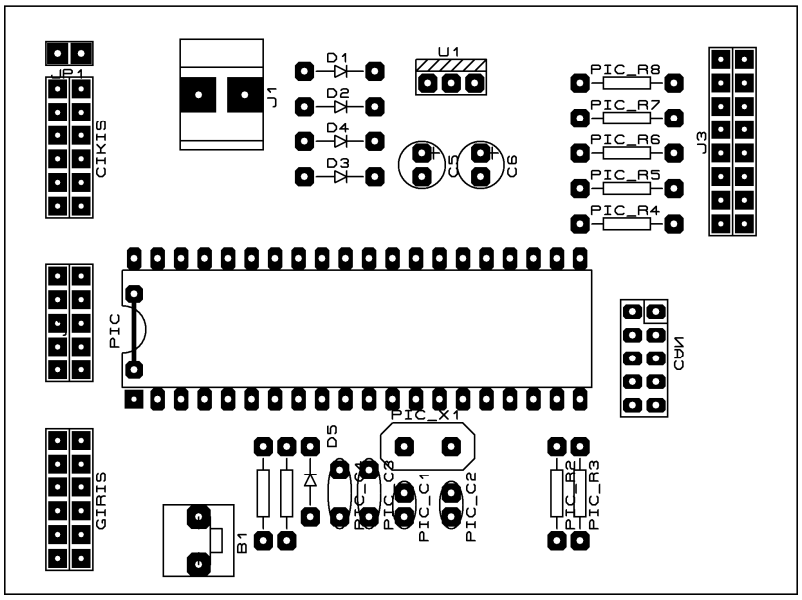
```
116         if ((PortA.1=1) and (RxTx_data[3]=Tx_ID)) then
117             RxTx_data[3]=$00
118             Veri_Gon
119         end if
120         if PortA.1=0 then
121             RxTx_data[3]=Tx_ID
122             Veri_Gon
123         end if
124         if PortA.2=1 then
125             RxTx_data[3]=$FF
126             Veri_Gon
127         end if
128         if PortA.3=1 then
129             RxTx_data[3]=$F0
130             Veri_Gon
131         end if
132         if ((RxTx_data[3]<>$F0) and (RxTx_data[3]<>$FF) and (RxTx_data[3]<>$00)) then
133             Delay_ms(300)
134             KAT=RxTx_data[3]
135             Kat_Yaz
136             Delay_ms(300)
137             KAT=255
138             Kat_Yaz
139             KAT=RxTx_data[0]
140             Kat_Yaz
141         end if
142 wend
143 end.
```

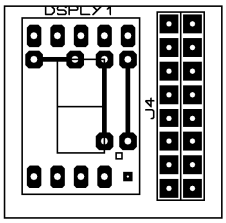
EK-F

Kat Kontrol Kartının Baskılı Devresi ve Yerleşim Planı



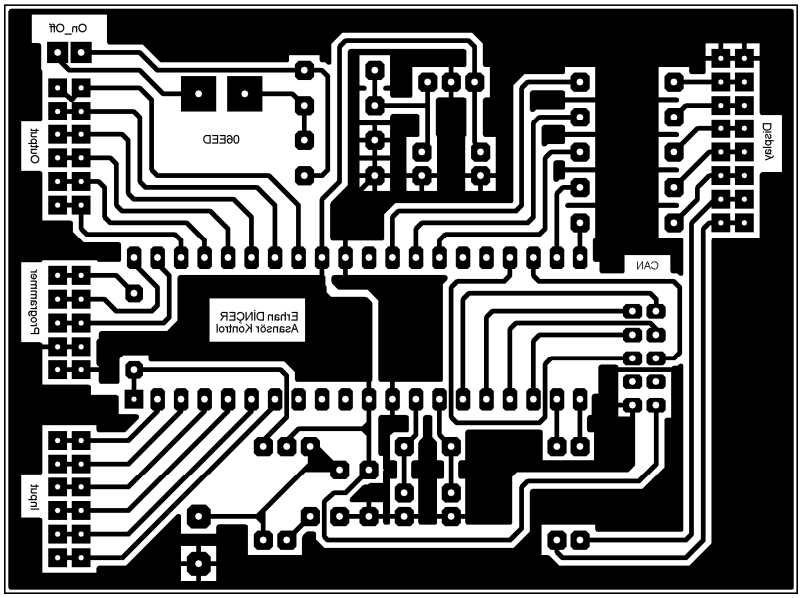


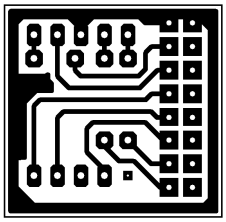


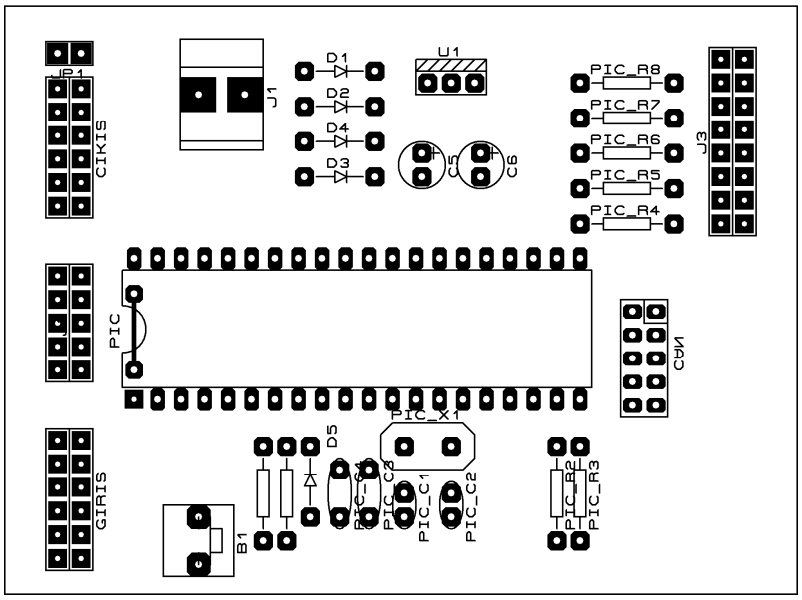


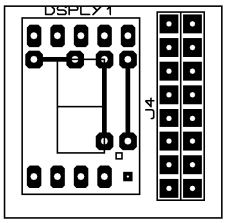
EK-G

Kabin Kontrol Kartının Baskılı Devresi ve Yerleşim Planı



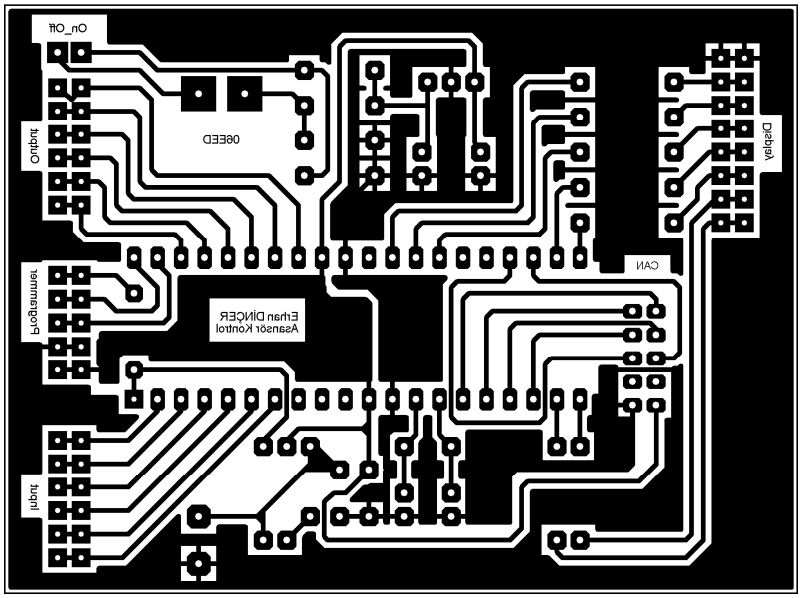


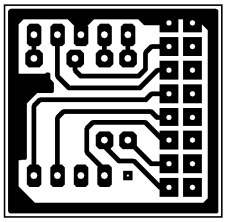


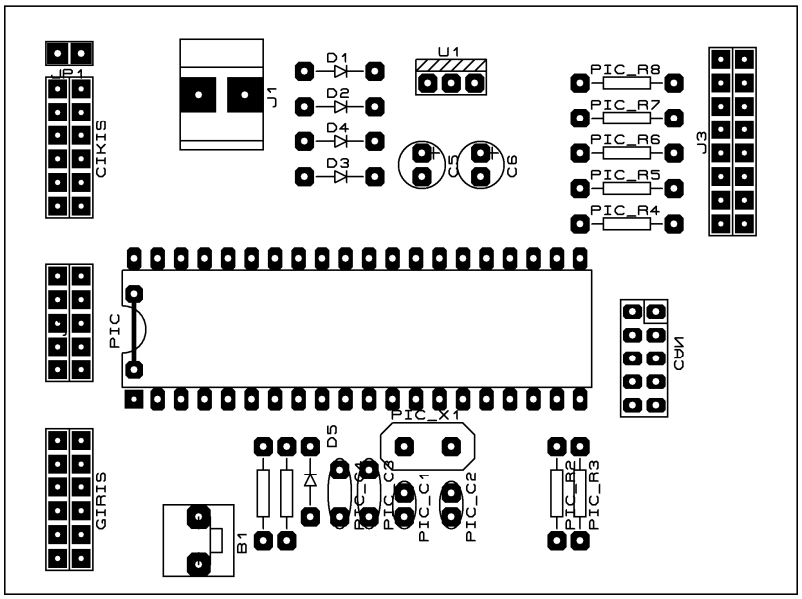


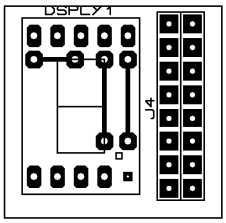
EK-H

Ana Kontrol Kartının Baskılı Devresi ve Yerleşim Planı









EK-I

CAN Modülü Kartının Baskılı Devresi ve Yerleşim Planı