KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MEKATRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HATA SİMÜLASYONLU CAN-BUS EĞİTİM SETİ GELİŞTİRİLMESİ

ÖZGÜR GÜÇLÜ

KOCAELİ 2018

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MEKATRONİK MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HATA SİMÜLASYONLU CAN-BUS EĞİTİM SETİ GELİŞTİRİLMESİ

ÖZGÜR GÜÇLÜ

Dr.Öğr.Üyesi Selçuk KİZİR Danışman,	Kocaeli Üniversitesi	.....
Doç.Dr. Cüneyt BAYILMIŞ Jüri Üyesi,	Sakarya Üniversitesi	.....
Doç.Dr. Kerem KÜÇÜK Jüri Üyesi,	Kocaeli Üniversitesi	.....

Tezin Savunulduğu Tarih: 05.04.2018

ÖNSÖZ VE TEŞEKKÜR

CAN haberleşmesi içerisinde bulunduğum otomotiv endüstrisi başta olmak üzere endüstride birden fazla alanda yaygın olarak kullanılmaktadır. Endüstri 4.0 ile birlikte akıllı sistemlerin daha fazla yaygınlaşması ve bunlarla birlikte CAN haberleşmesinin kullanımının da artması beklenmektedir.

Bu bağlamda, bana bu konuda çalışma ve kendimi geliştirme imkanı veren ve çalışma boyunca fikir ve yönlendirmeleriyle destek olan değerli hocam Dr. Öğr. Üyesi Selçuk Kizir'e teşekkürlerimi sunarım. Bununla birlikte hayatım boyunca olduğu gibi bu tez çalışması süresince de beni her anlamda destekleyen aileme sonsuz minnet duygularımı sunarım. Son olarak tez çalışması süresince anlayış gösteren ve beni destekleyen yöneticilerim ve çalışma arkadaşlarıma teşekkür ederim.

Nisan-2018 Özgür GÜÇLÜ

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

AC : Alternative Current (Alternatif Akım)
BRP : Baud Rate Prescaler (İletişim Hızı Ön Sayacı)
BTR : Bit Timing Register (Bit Zamanlama Saklayıcısı)
CAN : Controller Area Network (Kontrol Alan Ağı)
CRC : Cyclic Redundancy Check (Çevrimiçi Fazlalık Sınaması)
CSMA : Carrier Sense Multiple Access (Taşıyıcı Duyarlı Çoklu Erişim)
DC : Direct Current (Doğru Akım)
DLC : Data Length Code (Data Uzunluk Kodu)
DPLL : Digital Phase Lock Loop (Sayısal Faz Kilitlemeli Devre)
ESD : Electrostatic Discharge (Elektrostatik Boşalma)
IEEE : The Instute of Electrical and Electronics Engineers (Elektrik ve Elektronik Mühendisleri Enstitüsü)
ISO : International Organization for Standardization (Uluslararası Standartlar Teşkilatı)
OBD : On Board Diagnostics (Göstergeden Arıza Teşhisi)
OSI : Open Systems Interconnection (Açık Sistem Bağlantısı)
PCB : Printed Circuit Board (Baskı Devre Kartı)
REC : Receive Error Counter (Alınan Hata Sayacı)
RF : Radio Frequency (Radyo Frekansı)
RTR : Remote Transmit Request (Uzak İletim İsteği)
SAE : Society of Automotive Engineers (Otomotiv Mühendisliği Topluluğu)
SJW : Syncronization Jump Width (Senkronizasyon Atlama Aralığı)
SOF : Start Of Frame (Çerçeve Başlangıcı)
TEC : Transmit Error Counter (Gönderilen Hata Sayacı)
UDS : Unified Diagnostic Services (Birleşik Hata Ayıklama Servisi)

HATA SİMÜLASYONLU CAN-BUS EĞİTİM SETİ GELİŞTİRİLMESİ

ÖZET

Kontrol Alan Ağı (Controller Area Network ,CAN) yüksek güvenilirliğe sahip endüstriyel bir seri haberleşme protokolüdür. CAN haberleşmesi geliştirildiği günden günümüze kadar endüstride bir çok alanda kullanılmış ve kullanılmaya devam etmektedir. Özellikle otomotiv sektöründe çok fazla tercih edilen CAN haberleşmesinin güvenli ve hataya dayanıklı olması olumlu yanları olarak belirtilirken, özellikle iletim hızının sınırlı olması ve hız arttıkça fiziksel gerekliliklerinin değişmesi olumsuz yanları arasında gösterilebilir. CAN hatlarında yüksek hızlara çıkıldığında izin verilen maksimum hat uzunlukları eş oranda düşmektedir. İlerleyen yıllarda Endüstri 4.0 hareketi ile birlikte CAN haberleşmesinin daha da fazla yaygınlaşması ve CAN FD ile birlikte endüstrinin farklı alanlarında kullanılması beklenmektedir.

Bu tez çalışması kapsamında CAN haberleşmesinin tarihinden ve çalışma prensibinden detaylı olarak bahsedilecek ve örneklerle açıklanacaktır. Devamında CAN haberleşmesi üzerinde çalışmaya imkan tanıyacak ve bilgisayar üzerinde çalışan bir arayüze sahip olan bir eğitim seti tasarlanacak ve hem bilgisayar ortamındaki simülasyonlar hem de gerçek endüstriyel uygulamalardan alınan bir örnek üzerinden tasarlanan bu set yardımıyla daha fazla öğrencinin CAN haberleşmesi üzerinde çalışmasına ve hata yapıları, hat yoğunluğu ve mesaj yapısı gibi önemli konularda kendini geliştirmesine imkan tanınacaktır.

Anahtar Kelimeler: CAN, Endüstri 4.0, Endüstriyel Haberleşme Protokolü, Kontrol Alan Ağı.

DEVELOPMENT OF CAN-BUS TRAINING SET WITH ERROR SIMULATION

ABSTRACT

Controller Area Network (CAN), is an industrial communication protocol with high security standards. CAN communication is being used from the day it developed till today in lots of different areas within industry. In addition to being used mainly in automotive industry, the strong ways of CAN communication are shown as being secure and fault tolerant. On the other hand, some negative ways are the communication speed limitation due to physical necessities of high speed CAN lines. In following years, usage of CAN communication is expected to become higher along with CAN FD within the scope of Industry 4.0 revolution.

In this study, the history and significance of CAN communication will be explained through examples. Additionally, a training set will be designed with a computer interface, several simulation structures on computer and 3 different real CAN modules that will be created referencing real time automotive CAN-BUS systems. This aim of creating this set is to allow more students to study and learn CAN communication including bus load concept, CAN errors with message structures and contribute to related studies.

Anahtar Kelimeler: CAN, Industry 4.0, Industrial Communication Protocol, Controller Area Network.

GİRİŞ

Teknolojide yaygın olarak kullanılan seri haberleşme biçimlerinden biri olan Kontrol Alan Ağı (CAN) haberleşmesi 1986 yılında tanıdıldığı günden günümüze kadar pek çok alanda kullanılmış ve başta otomotiv endüstrisi olmak üzere kullanılmaya devam etmektedir [1]. Kullanılan akıllı sistemlerin sayısının artması ve bununla birlikte gelen maliyetler yeni ve pratik haberleşme yöntemlerinin kullanılmasını zorunlu kılmaktadır.

Endüstriyel sistemlerde söz konusu sistemin çok kısa sürelerde bile olsa çalışmayı bırakması maddi anlamda ciddi kayıplara yol açabildiğinden ötürü hataya dayanıklılık en önemli parametre olarak görülmektedir. CAN haberleşmesinin hataya dayanıklı olmasının yanında kendine ait endüstriyel standartları bulunduğu için varolan bir sisteme entegre edilmesinin kolay olması da önemli bir avantajıdır.

Bu tez çalışması kapsamında CAN haberleşmesinin avantajlarından bahsedilmiş, özellikleri ve çalışma prensipleri, hataya dayanıklı olmasını sağlayan sistemlere vurgu yapılarak detaylı şekilde açıklanmıştır. CAN haberleşmesinin söz konusu avantajlarının görülmesi ve daha fazla araştırmacı tarafından üzerinde çalışılması hedeflenmiş ve bu hedefe ulaşmak amacıyla bir eğitim seti tasarımı yapılmıştır. Eğitim seti hem CAN haberleşmesi hakkında bilgiler ve öğretici simülasyonlar içermekte hem de tez içerisinde bahsedilen özelliklerde çalışan bir CAN haberleşmesini gerçekleştirmekte ve dışarıdan kontrol edilip yönlendirilebilmektedir.

1. GENEL BİLGİLER

1.1. CAN BUS Sisteminin Tarihçesi

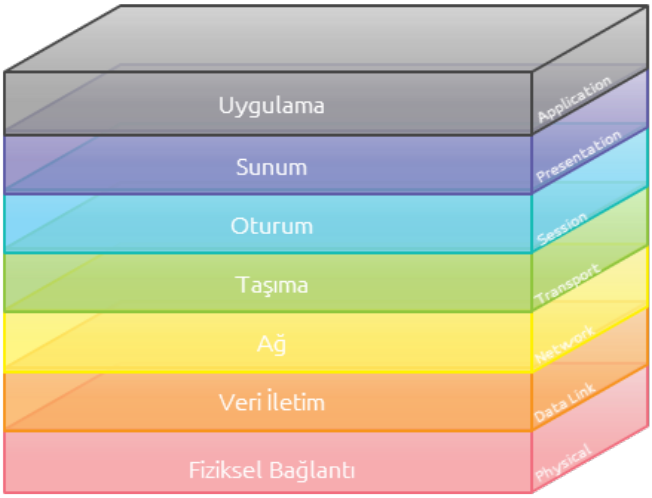
CAN BUS Robert Bosch tarafından 1983 yılında geliştirilmeye başlanmıştır. 1986 yılında Detroit'te yapılan SAE konferansında ise dünyaya tanıtılmıştır [1]. Intel ve Philips tarafından geliştirilen ilk CAN mikrokontrolörlerinin 1987 yılında piyasaya çıkmasının ardından 1988 yılında sunulan BMW 8 serisi araçlar CAN ile çalışan haberleşme sisteminin otomotiv endüstrisindeki ilk örneği olarak karşımıza çıkmaktadır.

CAN için Bosch tarafından bir çok spesifikasyon dökümanı yayınlanmış olsa da en şu anda 1991 senesinde yayınlanan CAN 2.0 spesifikasyonu kabul edilmekte ve kullanılmaktadır. Bu spesifikasyon iki kısımdan oluşmakta ve 11 bitlik CAN 2.0A ile 29 bitlik CAN 2.0B yapılarını anlatmaktadır [2].

1993 yılında CAN için Uluslararası Standartlar Teşkilatı (ISO) tarafından ISO 11898 adında bir standart yayınlanmıştır. Bu standart daha sonra farklı özellikleri içeren 3 parçaya ayrılarak güncellenmiştir ve günümüzde hala kullanılmaktadır.

1.2. CAN BUS Sisteminin Mimarisi

CAN ISO tarafından tanımlanmış olan OSI modelini kullanmaktadır. OSI modeli uygulama, sunum, oturum, ulaşım, ağ, veri bağlantısı ve donanım olmak üzere 7 katmandan oluşmaktadır [3]. Standart CAN mimarisinde veri bağlantısı ve donanım katmanları sağlanmakta ve bunların üzerine uygulama katmanı oluşturularak kullanılmaktadır. Diğer 4 katman (sunum, oturum, ulaşım ve ağ) CANOpen veya SAE J1939 gibi üst seviye CAN protokolleri ile belirlenmekte ve değişkenlik göstermektedir.



Şekil 1.1. OSI mimarisi [4]

Şekil 1.1'de OSI mimarisi şematik olarak gösterilmiştir. Fiziksel katman çoğunlukla haberleşmeyi sağlayan fiziksel parçaların elektriksel ve fiziksel etkileşiminin tanımlandığı ve bit çözümlemesinin yapıldığı katmandır.

Veri iletim katmanı ise fiziksel katmanın üzerine kurulan ve bu fiziksel altyapıyı kullanarak veri transferini sağlayan katmandır. Bu katman bazı kaynaklarda medya erişim ve mantıksal bağlantı katmanı olmak üzere ikiye ayrılarak incelenmektedir [4]. Bu şekilde bakıldığında medya erişim katmanı haberleşmeyi düzenlerken mantıksal bağlantı katmanı oluşabilecek hataları yakalamaya ve gidermeye çalışır. CAN sisteminin hataya dayanıklı olması bu katmanda bulunan yapılarla doğrudan ilgilidir.

Uygulama katmanı ise CAN mimarisinden bağımsız olarak düşünülebilir. Bu katmandaki herşey söz konusu uygulamaya özeldir ve CAN kullanılarak yapılacak işin tanımlandığı ve kaynak yönetiminin yapıldığı katmandır. Kullanılan CAN protokolüne göre sunum, oturum, ulaşım ve ağ katmanları ile zaman zaman iç içe geçebilmekte ve protokollerin getirdiği kısıtlamalardan ötürü birbirini etkilemektedir.

1.3. CAN BUS Sisteminin Avantajları ve Kullanım Sebebi

CAN kullanmanın en önemli avantajları, fiziksel bağlantıyı sayıca azaltarak kablolamayı kolaylaştırması ve farklı hata modlarına dayanıklı olmasıdır.

Aynı hat üzerinde birden fazla modülün bulunması prensibine dayandığı için CAN haberleşmesi fiziksel donanım gerekliliğini ciddi anlamda düşürmektedir. Örneğin,

otomotiv alanında 5 veya daha fazla modülün haberleşmesini 50 den fazla bağlantı yerine sadece iki tane kablo üzerinden sağlamak hem dizaynı kolaştırmakta hem de maliyetleri ciddi anlamda düşürmektedir.

CAN haberleşmesinin güvenli olmasının en önemli sebebi bir modül sıkıntı yaşadığında diğer modüllerin haberleşmeye kesintisiz devam edebiliyor olmasıdır. Aynı zamanda hat yoğunluğundan dolayı bir mesaj gönderilmez veya atlanır ise aynı mesajın tekrar gönderilmesi yoluyla hataların önüne geçilmektedir. Bununla birlikte, tanımlı protokoller kullanıldığında yakalanan herhangi bir hatayı görüntülemek ve gidermek oldukça basit olmaktadır [5].

CAN haberleşmesinin günümüzdeki avantajlarından biri de önceliklendirmeyi ayarlamanın basit olmasıdır. Gelişen teknoloji ile birlikte CAN hatlarında bulunan modüllerin ve dolayısıyla mesajların sayısı gittikçe artmaktadır. Bu sebepten ötürü önceliklendirme de her geçen gün daha önemli olmaktadır [6]. CAN haberleşmesinde dominant ve resesif bitler ile mesaj ID numaralarının ayarlanması yoluyla önceliklendirme yapılabilmektedir.

CAN günümüzde özellikle otomotiv sektöründe oldukça yaygın hale geldiğinden dolayı birçok üretici mikroişlemcilerin içerisinde hem fiziksel katmanı hem de veri bağlantısı katmanını hazır olarak dizayn etmekte ve sunmaktadır. Bu da kullanımını kolaylaştırmakta ve yapılan işleri standart hale getirmektedir.

1.4. BUS Hızları ve Bilinen CAN Protokolleri

CAN hatları için bilinen ve kullanılan birden fazla veri transfer hızı bulunmaktadır. 125 kbps, 250 kbps ve 500 kbps en sık kullanılan hızlardır. Bununla birlikte CAN haberleşmesinin üst limiti olarak bilinen 1000 kbps de nadir durumlarda kullanılmaktadır [2]. 1000 kbps olan hatlar sürekli haberleşme bulunmayan CAN hatlarıdır. Örneğin, bir modülün içerisine yazılım yüklenmesi veya 10 günde bir sadece bir kez yapılacak bir haberleşme için bu hız tercih edilmektedir. Özellikle yazılım yükleme işlemlerinde hattın hızlı olması bütün süreci hızlandırdığı için verimli olmaktadır. Standart hızlar, medium-speed CAN olarak tanımlanan 125 kbps ve high-speed CAN olarak tanımlanan 500 kbps olarak kullanılmaktadır. 250 kbps ise SAE

J1939 gibi bazı özel protokollerde tercih edilmekle birlikte 125 ve 500 kbps kadar yaygın bir kullanımı yoktur (Tablo 1.1).

Tablo 1. 1. CAN haberleşme hızları

Bus Hızı	Kullanım Hızı
125 kbps	Orta hızlı ve güvenilir uygulamalar
250 kbps	Özel uygulamalar (Ör. J1939)
500 kbps	Standart endüstriyel uygulamalar
1000 kbps	Kısa süreli kullanılan haberleşmeler

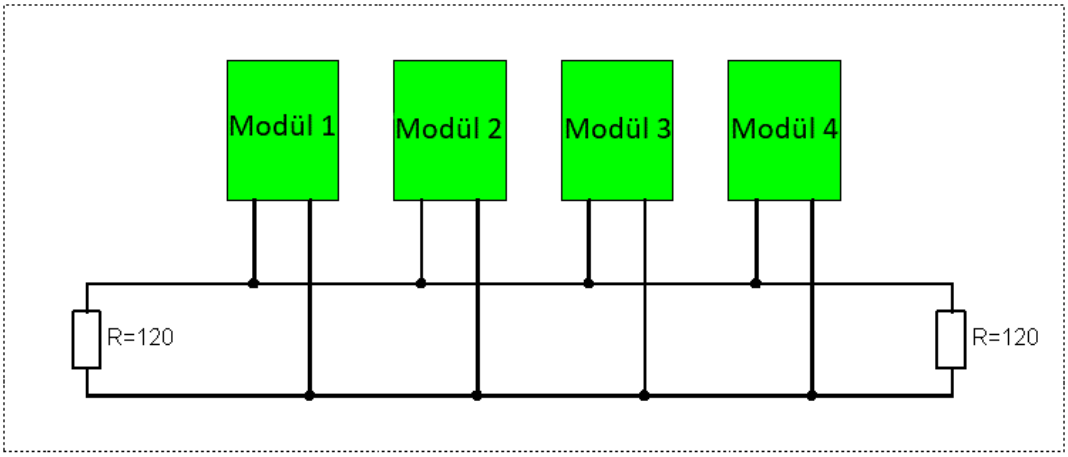
Bilinen ve sıklıkla kullanılan bazı CAN protokolleri; CANOpen, UDS, SAE J1939, ISO-TP ve DeviceNet protokolleridir. Bunlar çoğunlukla kullanım alanlarına göre ayrılmış ve özelleşmiş yapılardır [7]. Örneğin CANOpen ve DeviceNet çoğunlukla endüstriyel sistemlerde kullanılmakta, SAE J1939 ağır ticari araçlar ve otobüsler özelinde otomotiv sektöründe tercih edilmekte, UDS ise diagnostik amaçlı kullanılmaktadır.

Protokoller arasındaki farklılıklar, mesaj isimlendirmesi, modüllerin adreslenmesi gibi özellikleri barındırmaktadır. Örneğin, SAE J1939 standardını açtığınızda otomotiv sektöründe karşımıza çıkan motor kontrol modülü veya fren modülü gibi standart parçaların ve onların göndereceği ve alacağı mesajların ne şekilde isimlendirileceği ve gönderileceği belirlenmiştir. Bu sayede bu protokolü kullanan tüm araçların markalardan bağımsız olarak ortak bir strateji izlemesi sağlanmaktadır.

1.5. CAN BUS Sistemi ile İlgili Temel Bilgiler ve Çalışma Prensibi

1.5.1. Veri yolu ve bağlantı noktaları

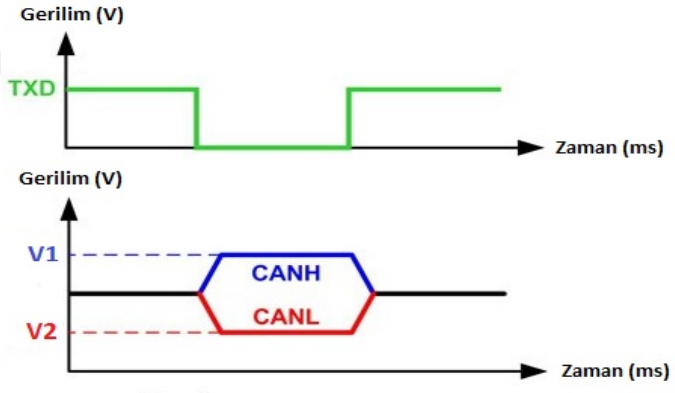
CAN haberleşmesinde veri yolu iki fiziksel bağlantıdan oluşmaktadır. Bunlar CAN-H ve CAN-L olarak adlandırılan iki kablodur. Tüm modüller bu iki kabloya bağlanarak hep birlikte haberleşir. Bağlantı noktaları birden fazla sayıda olabilir ve aynı hat üzerinde bulunduklarından dolayı hepsi birbirine mesaj gönderip alabilirler [8]. CAN haberleşmesinde veri yolunun iki ucu 120 ohm dirençlerle korunmaktadır (Şekil 1.2).



Şekil 1.2. CAN Hattı veri yolu

1.5.2. Mesaj önceliklendirilmesi

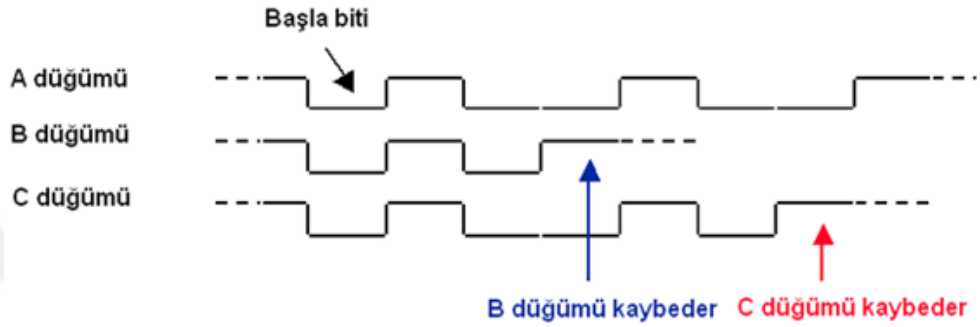
CAN hattının lojik seviyesi 1 veya 0 olabilmektedir. Bu yapıda 1 resesif 0 ise dominant bit olarak adlandırılır. Bunun sebebi 1 ve 0 aynı anda hatta basıldığında 0 bitinin 1'e baskın gelmesidir (Şekil 1.3).



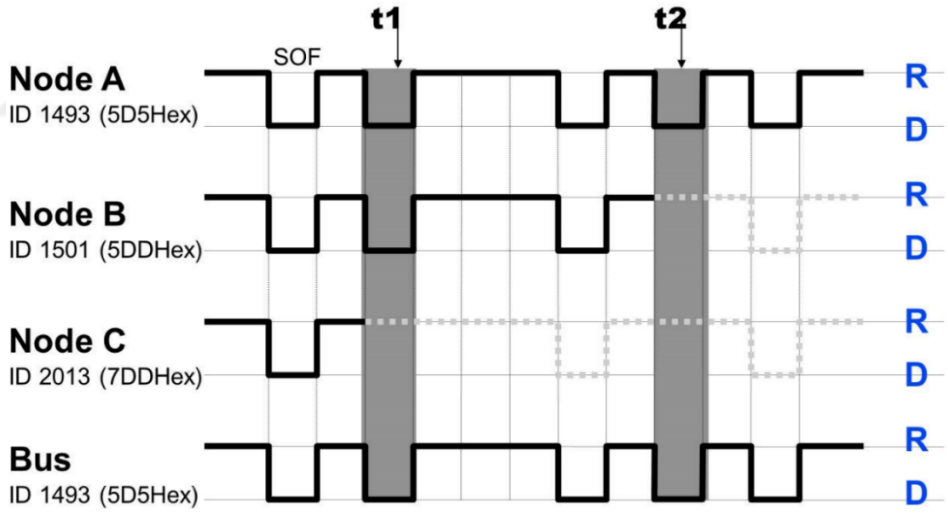
Şekil 1.3. CAN dominant ve resesif sinyal yapısı

CAN hattı 1 ve 0 lojik seviyeleri arasında belirlenen hızda değişim göstererek haberleşme sağlar. 0 bitinin 1'e üstün gelmesi özellikle mesaj önceliklendirilmesinde ve hataların yakalanmasında fayda sağlamaktadır. Bu sayede mesaj numaraları küçük olan mesajlar daha öncelikli olurlar [9]. Bütün modüller sürekli olarak hattı dinlemektedir. Bir modül hattın boş olduğu anı yakaladığında ilgili mesajı gönderir. Gönderilen mesaj içerisinde kime gönderildiği de bulunmaktadır. Alıcı modül bu bilgiyi kullanarak hatta basılan bu mesajı yakalar ve kullanır.

CAN hattına aynı anda mesajlar basıldığında 0 dominant bitine sahip olan mesajlar diğerlerine üstün gelerek hatta kalır. Resesif bite sahip olan mesajlar ise geride kalır ve hattın boşalmasını bekleyip sonrasında tekrar gönderilirler. Bu yapı sayesinde mesajlar bir kere resesif kaldığında iptal olmayıp tekrar gönderilmekte, bu da hattın daha güvenli olmasını sağlamaktadır [10]. CAN mesaj önceliklendirilmesine ait iki örnek Şekil 1.4 ve Şekil 1.5'de gösterilmiştir.



Şekil 1.4. CAN mesaj önceliklendirilmesi örnek 1 [10]



Şekil 1.5. CAN mesaj önceliklendirilmesi örnek 2 [10]

1.5.3. Multimaster çalışma yapısı

CAN haberleşmesinde diğer modülleri yöneten tek bir dominant(master) modül yoktur. Hattın o anki durumuna ve mesajların önceliklerine göre her modül sırası geldiğinde hatta mesaj verip master modül görevi görebilir [11]. Fiziksel olarak alıcı ve verici birbirinden bağımsız olmasına rağmen yukarıda bahsedilen önceliklendirme yapısı sayesinde resesif bir biti hatta gönderecek olan modül dominant biti gördüğünde

hattı ona bırakır ve sırasını bekler. Bu sayede hattı sürekli dinleyen tüm modüller bir arada haberleşebilir.

Bu yapı özellikle bir modülün arıza yapması durumunda diğer modüllerin sorunsuz çalışabilmesi açısından önemlidir. Hattın yapısı tek bir master modüle bağlı olmadığından dolayı bir modülün arızasından dolayı diğer modülleri de etkileyecek bir hata modu oluşmasının önüne geçilmektedir.

1.6. CAN Mesaj Yapıları

1.6.1. Mesaj çerçevesi ve istek çerçevesi kavramları

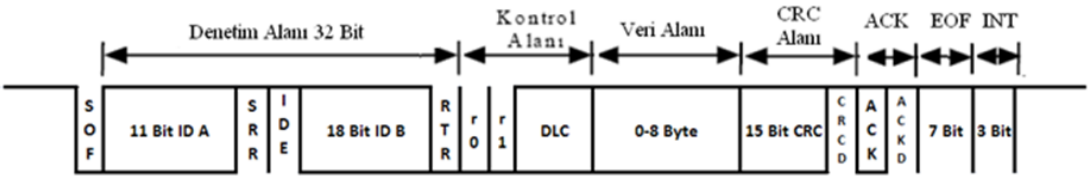
CAN hattına birden farklı yapı göndermek ve almak mümkündür. Bunlardan bir tanesi standart CAN mesajı olarak da bilinen message frame yapısıdır. Bu yapı protokolden protokole farklılık göstermekle birlikte bir denetim alanı ve bir veri alanı içerir. Denetim alanında özellikleri belirlenirken veri alanı ise gönderilmek istenen bilgiyi taşımaktadır.

Bazı durumlarda modüllerin hatta veri göndermek yerine istek yapmaları gerekir. Bu istekler çoğu zaman hattın boş olup olmadığını anlamak yada aktif olup olmadığını görmek içindir. Bu istek yapıları request frame olarak adlandırılır. Bu yapının normal mesajlardan farkı denetim alanında bir bitin dominant yerine resesif olması ve herhangi bir bilgi aktarımı yapma amacı olmadığından dolayı veri alanı içermiyor olmasıdır.

1.6.2. CAN 2.0A ve CAN 2.0B standartları

Bosch tarafından yayınlanan CAN standardında 2 farklı CAN standardı bulunmaktadır. Bunlar CAN 2.0A ve CAN 2.0B standartlarıdır [2]. Bu standartların temel farkı kullandıkları mesaj çerçeve yapısının farklı olmasıdır. CAN 2.0A standardında temel çerçeve yapısı kullanılırken CAN 2.0B genişletilmiş çerçeve yapısı kullanır. Bu çerçeveler arasındaki en temel fark mesaj ID yapısının 2.0A standardında 11 bit 2.0B standardında 29 bit olmasıdır (Şekil 1.6 ve Şekil 1.7).

Şekil 1.6. CAN 2.0 A mesaj yapısı [12]



Şekil 1.7. CAN 2.0 B mesaj yapısı [12]

Her çerçeve SOF (Start Of Frame) sinyali ile başlar. Bu sinyal 1 bitliktir dominanttır. Bunun ardından 12 bitlik denetim alanı gelmektedir. İlk 11 biti mesaj ID alanıdır ve bu alandaki ID değeri ile mesajlar etiketlenir. Denetim alanındaki son bit RTR diye adlandırılır ve özel anlamı vardır. Bu bit 0 (dominant) ise gönderilen çerçeve veri çerçevesidir ve veri alanında, ID alanında tanımlanan mesaja ait veri vardır. Bu bit 1 ise çerçeve istek çerçevesidir ve veri alanı yoktur. ID alanındaki ilk 7 bit ardışıl olarak resesif olamaz aksi taktirde hata mesajı olarak algılanacaktır [12].

Denetim alanından sonra kontrol alanı gelmektedir. Bu alanın ilk biti IDE diye isimlendirilir ve bu çerçevenin 11 bitlik ID alanına sahip 2.0A çerçevesi olduğunu belirten dominant bir bittir. Bu bitin ardından bir bitlik kullanılmayan rezerve alan gelmektedir.Daha sonra 4 bitlik DLC diye isimlendirilen bir alan gelir. DLC alanı gönderilen verinin kaç byte olduğunu söyler. Kontol alanını veri alanı takip etmektedir. Veri alanı en fazla 8 byte olabilmektedir ve mesaj ile gönderilmek istenilen bilginin bulunduğu asıl bölümdür. Veri alanını CRC alanı takip eder. Bu alan 16 bitliktir ve 15 bitlik CRC bilgisi ile resesif CRC Delimiter bitinden oluşmaktadır. CRC alanı gönderilen SOF alanından CRC alanına kadar gönderilen verinin doğru olup olmadığının anlaşılması için bir değerdir.Veriyi gönderen düğüm veri üzerinde bir takıp işlemler yaparak 15 bitlik CRC değerini hesaplar ve çerçeveye ekler. Alan düğüm veriyi aldığında göndericinin yaptığı işlemler ile aynı işlemleri yapar ve CRC yi tekrar hesaplar. Alınan ve gönderilen CRC tutarlı ise veri doğru gönderilmiştir.Alıcı

düğümlerden en az 1 tanesi bile veriyi yanlış aldıysa veri tekrar gönderilmelidir. [13] CRC alanını ACK alanı takip eder.Bu alan 2 bitliktir.İlk bitini gönderici çekinik olarak gönderir.Eğer veri en az bir alıcı tarafından doğru alınmışsa alıcı yola baskın biti yazar. Böylece gönderici mesajın en az bir alıcı tarfından alındığını anlar. Eğer gönderici dominant biti okuyamassa ACK işaretinden kaynaklı bir hata olduğuna kanaat getirir ve veriyi tekrar yollar.Bu alanın ikinci biti ise ACK delimiter olarak adlandırılır ve resesiftir. Daha sonra çerçevenin sonlandırıldığını belirten 7 bitlik EOF alanı gelir. Bu alandaki bitler resesiftir. Daha sonra ise çerçeveler arasında boşluk bırakmak amacıyla 3 bitlik INT alanı gelmektedir ve bitleri resesiftir.

CAN 2.0B de mesaj ID si 29 bittir.Geriye uyumluluk açısından CAN2.0B geliştirilirken 2.0A göz önünde bulundurularak geliştirilmiştir. İki protokolde aynı yolda çalışabilmektedir fakat 2.0A düğümlerine 29 bitlik ID li mesajlar gönderilmemelidir. Genişletilmiş çerçevede dominant SOF ile başlar ardından 2.0A da olduğu gibi 11 bitlik IDA alanı gelir. Ardından 2.0A daki dominant RTR biti yerine resesif SRR biti gelmektedir.Ardından 2.0A daki ofsete denk gelecek şekilder IDE biti gelir.Tek farkı 2.0B de bu bitin resesif olmasıdır çünkü 29 bitlik ID ye sahip mesajlar iletilmektedir.IDE bitinden sonra 18bitlik ikinci ID B alanı gelir.Ardından dominant RTR biti gelerek bu çerçevenin veri çerçevesi olduğunu belirtir.Kontrol alanının ilk iki biti rezerve dir ve kullanılmaz. Son dört bit DLC alanını oluşturur ve gönderilen verinin kaç byte olduğunu söyler. Geri kalan kısım 2.0A ile aynıdır [14].

1.6.3. Hat yükü kavramı

Can haberleşmesinde gönderilen ve alınan her mesaj hatta belirli bir yük bindirir. Hattın hızına göre belirli bir kapasitesi vardır. Bu kapasite bir saniyede kaç bitlik veri transferi yapabildiğine göre ölçülür. Bir mesajın kaç bitlik veri taşıdığı ve ilgili çerçevenin kaç bitten oluştuğu bilindiği için bu yoğunluk hesaplanabilir [15].

Eğer CAN hattının yoğunluğu belirli değerlerin üzerine çıkarsa düşük öncelikli bazı mesajlar her zaman baskılanacağından dolayı hiç bir zaman gönderilemezler veya geç gönderilirler. Böyle bir durumla karşılaşıldığında sistemin çalışma hızı ve güvenilirliği düşecektir ve zaman gecikme hatalarına sebebiyet verme ihtimali artacaktır. Görülebilecek sıkıntılar hatalar bölümünde detaylarıyla açıklanacaktır. Bu tip

yoğunlukların önüne geçmek adına bir hattın yükünün en fazla %80, ideal durumda ise %50-60 aralığında olması istenir.

Hat Yoğunluğunu hesaplamak için her bir mesajın hatta oluşturduğu yoğunluğu ayrı ayrı hesaplamak gereklidir [16]. Sonrasında hat yoğunluğu bu değerlerin toplamına denk gelecektir. Bir mesajı N tane modülün aldığını düşündüğümüzde tek bir mesajın hatta oluşturduğu yoğunluk mesajın basılması ve dinleyen her bir modülün buna verdiği yanıt bir araya getirilerek hesaplanır.

$$t_{mesaj} = t_{talep} + t_{cevap1} + t_{cevap2} + \ldots + t_{cevapn} \quad (1.1)$$

Sonrasında bütün hattın yoğunluğu mesajların oluşturduğu yoğunluklar toplanarak bulunabilir. N tane mesaj olduğu düşünülür ise hat yoğunluğu aşağıdaki gibi hesaplanabilir.

$$\text{Hat Yoğunluğu} = t_{mesaj1} + t_{mesaj2} + t_{mesaj3} + \ldots + t_{mesajn} \quad (1.2)$$

1.7. Bit Süresi ve BTR Hesabı

Bir bit senkronizasyon segmenti (SYNC_SEG), yayılma segmenti (PROP_SEG), birinci faz segmenti(PHASE_SEG 1) ve ikinci faz segmenti (PHASE_SEG 2) olarak isimlendirilen 4 parçadan meydana gelmiştir. Bu parçaların toplam süresi bit süresini belirler.

$$t_{bit} = t_{syncseg} + t_{propseg} + t_{phaseseg1} + t_{phaseseg2} \quad (1.3)$$

1.7.1. Kuanta kavramı ve segmentler

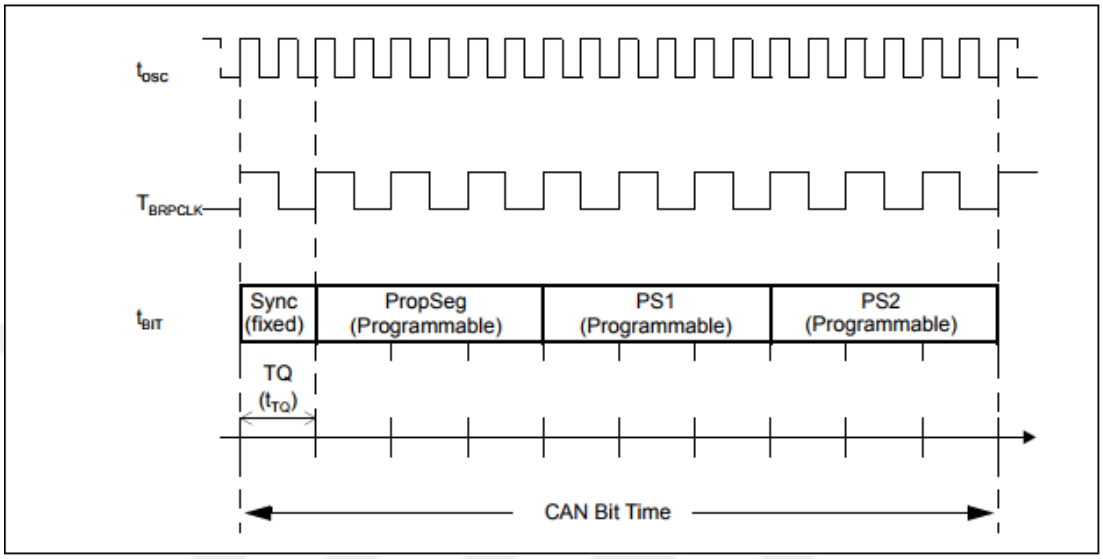
Bit zamanı hesabında birim zaman, kuanta adı verilen bir birim ile ölçülmektedir. Bir kuantanın ne kadarlık bir zaman dilimine karşılık geldiği kullanılan osilatörün frekansı ile alakalı bir konudur [17]. Temelde osilatör periyodunun iki katı bir kuanta olarak programlanır ancak osilatör frekansının yanı sıra BRP adı verilen programlanabilir bir yapının değiştirilmesi ile bir kuantanın karşılık geldiği zaman dilimi de ayarlanabilmektedir.

$$t_{kuanta} = 2 * BRP * t_{osi} = \frac{2*BRP}{f_{osi}} \quad (1.4)$$

t_{kuanta} = Bir Kuantanın Süresi

t_{osi} = Osilatör Periyodu

f_{osi} = Osilatör Frekansı



Şekil 1.8. CAN bit zaman hesaplaması [18]

Senkronizasyon segmenti sabit olarak 1 kuanta uzunluğunda olmaktadır. Her bitin en başında bulunur ve tüm modüllerin birbirine senkron çalıştığından emin olmak için kullanılır. CAN hattı üzerindeki tüm modüller aynı bit süresine sahip olmalıdır. Bunu sağlamak için hat üzerindeki tüm modüller hat durağan konumdayken bir SOF biti geldiğinde timerlarını senkronizasyon segmentine ayarlarlar. Böylece hepsi aynı konumda senkronize olmuş olurlar (Şekil 1.8).

Yayılma segmenti bus üzerinde gerçekleşecek gecikmeleri gidermek amacıyla kullanılmaktadır ve 1 ile 8 kuanta arası uzunluklarda olabilir. Gecikmeler örnekleme zamanının üzerinde bir miktara çıkarsa hat üzerinde önceliklendirme hataları görülebilir. Hiç gecikme olmadığı ideal durumda önceliklendirme dominant ve resesif bitlere göre yapılmaktadır ancak gecikme durumunda dominant bit göndermesi gereken bir modül gönderemez ve onun yerine başka bir modülün resesif biti önceliklendirilir ise bu sistemin tamamında sıkıntı ve gecikmeye sebep olabilmektedir [18].

Gecikmeler hata durumları dışında 2 temel sebepten gerçekleşmektedir. Bir tanesi göndericinin verici yapısında kaynaklanan gecikmeler, ikincisi ise alıca tarafındaki kıyaslayıcı devrede olan gecikmeler. Bu iki gecikme ve sinyalin hat üzerindeki iletim zamanı hesaba katılarak bu gecikmeyi önleyebilecek yayılma zamanı hesaplanabilmektedir.

$$t_{yay} = 2 * (t_{bus} + t_{dri} + t_{com}) \quad (1.5)$$

t_{yay} = Yayılma Zamanı

t_{bus} = İletim Süresi

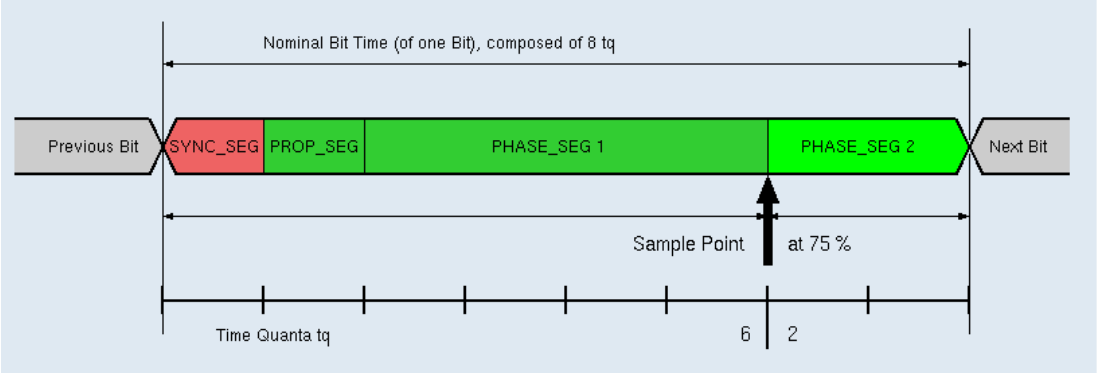
t_{dri} = Verici Gecikmesi

t_{com}= Kıyaslayıcı Gecikmesi

Birinci ve ikinci faz segmentleri bir bütün olarak düşünülebilirler. Örnekleme zamanına göre tek bir segmentin ikiye bölünmesi ile programlanırlar. Birinci faz 1 ve 8 kuanta arasında ikinci faz ise 2 ve 8 kuanta arasında değerler alabilmektedir. Bu segmetin ikiye bölecek yapı programlanırken dikkat edilmesi gereken bazı kurallar vardır. İkinci faz segmenti yeniden senkronizasyon esnasında değişikliğe uğrayabilir. Bu kurallar SJW kavramı ile birlikte örnekleme zamanı bölümünde detaylarıyla açıklanacaktır.

1.7.1. Örnekleme zamanı seçimi

Örnekleme noktası söz konusu bit üzerindeki lojik seviyenin alındığı ve yorumlandığı noktadır. Diğer bir deyişle bitin anlam kazandığı nokta olarak düşünülebilir [19]. Bu noktadan sonra ikinci faz segmenti ile belirli bir zaman gecikmesi sağlanır ve bir sonraki bite geçilir. Bitler arasında gecikmelerden dolayı oluşan kaymalar olmaması açısından örnekleme noktasının hangi noktada seçildiği ve faz segmentlerini hangi noktadan böldüğü önem taşımaktadır. Burada SJW kavramı da karşımıza çıkmaktadır (Şekil 1.9).



Şekil 1.9. CAN örnekleme zamanı gösterimi [18]

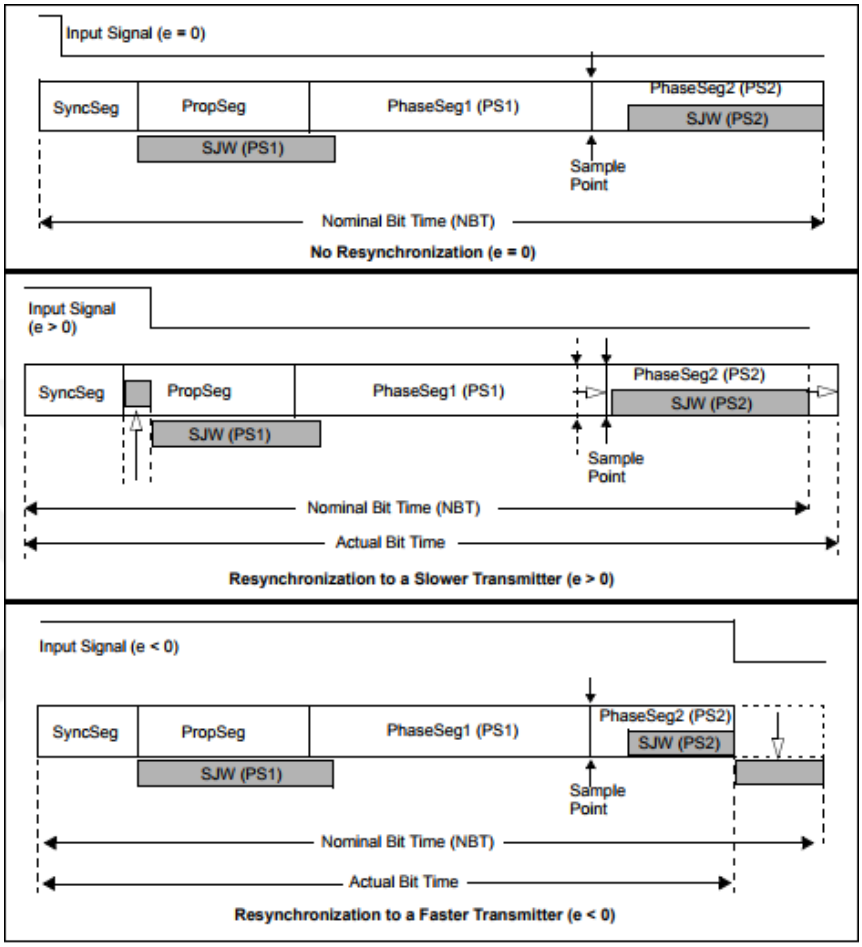
SJW bit zamanını yeniden senkronize etmek, diğer bir deyişle uzatmak veya kısaltmak için kullanılır [20]. Osilatörde oluşabilecek çok ufak zaman kaymalarından dolayı bazı durumlarda örnekleme zamanı istenilen noktanın dışında kalmaktadır. Bu durumlarda önceki bölümlerde bahsedilen senkronizasyonun senkronizasyon segmenti dışında tekrarlanması gerekmektedir. Bu tekrarlama işlemi belli kurallar çerçevesinde yapılabilmektedir.

DPLL (Digital Phase Lock Loop) adı verilen bir yapı kullanılarak örnekleme noktası önceden olması beklenen örnekleme noktası ile sürekli olarak karşılaştırılır ve arada bir fark var ise belirlenir [21]. Bu farka göre bit süresi uzatılmalı veya kısaltılmalı, işlem sonucunda örnekleme noktası aynı noktada kalabilmelidir. Bu işlem ikinci faz segmentinin uzaması veya kısalması ile mümkün olabilmektedir. Senkronizasyon aşağıdaki kurallar çerçevesinde gerçekleştirilmektedir:

- Sadece çekinikten baskın kenarlara geçişte, diğer bir deyişle sadece alçalan kenarlarda senkronizasyon yapılabilir.
- Bir bitlik sürede en fazla bir senkronizasyona izin verilmektedir.
- Bir önceki örnekleme noktasına göre farkı oluşturan noktadan hemen önceki kenarda senkronizasyon işlemi gerçekleşmelidir.
- Eğer oluşan faz hatasının mutlak değeri SJW değerine eşit veya büyük ise senkronizasyon sırasında yapılacak kayma en fazla SJW kadar olabilir.

Genellikle SJW değeri en hızlı ve en yavaş modüller arasındaki fark gözetilerek en kötü durum senaryosuna göre seçilir. Ne kadar büyük olursa sistem o kadar dayanıklı olacaktır. Aksi taktirde oluşacak gecikmeler SJW değerinden büyük olacağı

durumlarda senkronizasyon işlemi ile tam olarak giderilememiş olurlar. Aynı şekilde SJW değeri de ikinci faz segmentindeki değişikliği sınırladığı için ikinci faz segmentinden daha uzun olamaz (Şekil 1.10).



Şekil 1.10. SJW kullanımı [18]

1.8. BUS Hızının Belirlenmesi

CAN hattının hızı teorik anlamda bakıldığında hattın yoğunluğu ve bağlı olan modül sayısına göre hesaplanmalıdır. Modüllerin bastıkları mesaj sayılarına ve bu mesajların tekrarlanma zamanlarına göre busload %60 değerini geçmeyecek şekilde tasarlanan bir CAN hattı dayanıklı ve uygun olarak nitelendirilebilir. Bus hızı arttıkça bir saniyede gönderilebilen mesaj sayısı ve dolayısıyla hattın destekleyebileceği mesaj sayısı da artmaktadır.

Günümüzde kullanılan CAN yapılarında 125, 250 ve 500 kbps standart hızlar olarak belirlenmiş ve kullanılmaktadır. Kullanılan protokollere ve kullanım alanlarına göre

belirlenen hızlar, çok özelleşmiş bir tasarım yapılacağı durumlar dışında yeterli olmaktadır.

125 kbps hızlar az sayıda modülün olduğu CAN hatlarında tercih edilmektedir ve medium speed veya low speed CAN olarak isimlendirilirler. Low speed can hatları fiziksel katmanda farklılıklar bulundurur. Piyasada düşük hızdaki CAN hatları genellikle hataya dayanıklı olması istenilen sistemler için tercih edilmektedir. 250 ve 500 kbps CAN hatları ise high speed kategorisinde sınıflandırılmaktadır.

500 kbps bir CAN hattı için bir saniyede gönderilebilecek maksimum çerçeve sayısı 9000 civarlarındadır. Bu sayı 250 kbps bir CAN hattı için yarıya düşmektedir. Hat yoğunluğu değerini %60 dolaylarında tutmak istenildiğini hesaba katarsak 250 kbps endüstriyel uygulamalarda yetersiz kalmaktadır. Çoğunlukla otomotiv uygulamalarında, özellikle ağır ticari araçlarda kullanılan J1939 standardında 250 kbps tercih edilmektedir [22].

Bunların dışında bazı endüstriyel uygulamalarda CAN hatları fiziksel sınırlarına dayanarak 1000 kbps hızında kullanılmaktadır. Bu hıza çıkıldığında toplam hat uzunluğunun 30 metreden daha uzun olmaması gerekmektedir aksi taktirde hat bozuculara karşı zayıf duruma düşmektedir [17]. Bu tip yapılar güvenliğe ve dayanıklılığa önem verilen uygulamalarda tercih edilmez.

1.9. CAN Hataları

1.9.1. Hata çerçevesi ve hata durumu

CAN hattında bir hata oluştuğunda modüllerin oluşturduğu bir hata çerçevesi bulunmaktadır. Bu çerçeve özelleşmiş bir yapıdadır ve bit stuffing kuralını ihlal eder. Bu kural ihlal edildiği için tüm modüller hatta böyle bir çerçeve gördüklerinde bir hata olduğunu algılar ve buna göre davranırlar.

Bit stuffing kuralına göre çekinik veya baskın aynı tipten 5 ten fazla bit arka arkaya sıralanamaz [23]. Bir hata modu oluştuğunda ilgili modül hatta üst üste 6 adet dominant bit gönderir. Bu çerçeveyi fark eden diğer modüller de bir hata olduğunu algılarlar.

CAN haberleşmesinde tüm işlemciler TEC (Transmit Error Counter) ve REC (Received Error Counter) adı verilen iki adet özel register bulundururlar. Bu registerlar 8 bit büyüklüğündedir. Bir modül hata ürettiğinde kendi TEC registerını arttırır. Bu hatayı alan diğer tüm modüller ise REC registerlarının değerini arttırırlar. Bu registerlar tamamen dolana kadar, başka bir deyişle değerleri onaltılık tabanda 0xFF değerine eşit oluncaya kadar hatalara aldırmadan CAN hattı çalışmasına devam eder [24]. Bir sonraki denemede aynı mesaj gönderilebilirse ilgili TEC ve REC registerlarının değerleri azaltılır. Genellikle sadece 1 azaltılır. Arttırıldığı miktarda azaltılması da mümkündür ama bu yapılır ise CAN hattındaki bir hata tamamen elimine edilmiş olur bu da hattın güvenilirliğini düşürecektir. Eğer bu registerlardan herhangi biri 0xFF değerine ulaşacak olursa ilgili modül BUS OFF moduna geçer ve hatalı veya hatasız ayırt etmeden tüm mesajlarını basmayı ve hattı dinlemeyi bırakır. Üretilen modüllerin büyük bir kısmı hata moduna geçtikten sonra enerjisi kesilip geri verildiğinde hata modundan çıkar ve bu registerları sıfırdan başlatır. Ancak bazı modüller özellikle hataya hassas uygulamalarda kullanılıyorlar ise dışarıdan müdahale olmadan normal çalışma durumlarına geri getirilemezler.

1.9.2. Zaman gecikmesi hataları

Time out yapısı CAN spesifikasyonlarında yer almaz. Yazılımların içerisinde geliştirilen ve hata durumlarının önüne geçmek amacıyla kullanılan yapılardır [25]. Daha önce de bahsedildiği gibi CAN haberleşmesi esnasında bütün modüller hattı sürekli olarak dinler ve talep gönderirler. Bir modül talep ettiği ve beklediği bir mesajı, belirli bir süre boyunca veya belirli bir deneme sayısında alamaz ise time out hatası üretir veya kendini limp-home ismi verilen özel bir moda alır [26]. Bunun yapılmasındaki amaç kullanıcıyı bilgilendirmek ve daha ileri seviyede hatalara sebep olmadan önce sistemi koruma moduna almaktır. Time out hatalarının oluşma şekilleri, limitleri ve doğuracağı sonuçlar tamamen yazılım tarafından belirlenir. Bunun için bir standart yada spesifikasyon takip edilmemektedir.

Örneğin, bir aracın hareketi esnasında vites kutusunun arızalandığı durum ele alındığında aracın arızanın ardından uygun şekilde vites değiştirmesinin mümkün olmayacağı bilinmektedir. Bu durumda modülün hata üretmesi ve yukarıda bahsedildiği şekilde modülün TEC registerını doldurarak tamamen CAN hattından

uzaklaşması sürücüyü riske atacak bir durumdur. Bunun yaşanmaması için vites kutusundan sorumlu olan modül bir hata olma durumun önceden ayırt eder ve time out hatası üretir. Bu hatayı ürettiğinde aynı zamanda özel bir moda girer, bu mod ile birlikte sürücünün sürüşe devam etmesine izin verecek bir vitese geçerek o seviyede kalır ve daha sonra arabayı boş vitese getirir. Bu şekilde hem sürücüyü riske atmamış hem de arabanın aktarma organlarının zarar görmesini önlemiş olur.

Bu tip modlar faydalı olmakla birlikte moddan çıkma ve normale dönme durumlarının da çok düzgün tanımlanması gerekmektedir. Modüllerin basit hatalar sonucunda girdiği modlardan özel müdahale olmadan çıkamaması, endüstride üretimi yavaşlatacak, kullanıcı deneyimi ile alakalı sistemlerde ise olumsuz geri dönüşlere sebep olacaktır.

1.9.3. Hata önleme yapısı

CAN BUS haberleşmesinin en büyük avantajı oluşabilecek bir çok hataya dayanıklı ve dolayısıyla güvenilir bir sistem olmasıdır. CAN BUS birçok modülü aynı anda bünyesinde bulundurmakta ve bu modüllerin hepsi birbiriyle konuşabilmektedir. Böyle bir sistemin hataya oldukça açık olması beklenirken modüllerin birbiri ile karışmamasının en önemli sebebi ID yapısıdır. Her mesajın bir ID si bulunur ve bu ID nin belirli bir kısmı mesajı gönderen modülün hangi modül olduğunu söylemek amacıyla ayrılmıştır [27]. Bu şekilde bir mesajı tüm modüller dinlediği halde sadece ihtiyacı olan modül alabilir ve kimden geldiğini bilir.

CAN BUS haberleşmesinin güvenliğinde CSMA ismi verilen bir yapının da büyük payı vardır [28]. Bu yapı sayesinde tüm modüller hattı sürekli dinlerler. Hat boş olduğunda mesajlarını gönderirler. Bazı nadir durumlarda bir ünite mesaj gönderdiğinde, özellikle yolun uzunluğundan dolayı başka bir modül bu ünitenin gönderdiği mesajı algılayamaz ve yolu boş zannederek mesaj gönderir. Bu tip durumlar olduğunda iki modül de bu durumu algılar ve mesajı bir süre bekletirler. Daha sonra yolu tekrar kontrol ederek boş ise mesajlarını gönderirler. Bu şekilde mesaj çakışmaları ve üst üste binme sorunları giderilmiş olmaktadır.

Hatalara karşı bir başka önlem ise CRC yapısı ile sağlanmaktadır [29]. Cyclic Redundancy Check adı verilen bu yapı her bir mesaj çerçevesinde yer alır ve 15 bitlik

bir bölüm ile 1 bitlik limitleme bölümünden meydana gelir. Her bir mesajın özelinde bir CRC değeri hesaplanmaktadır. Gönderici bu değer ile birlikte mesajı gönderdiğinde alıcı modül kendi hesapladığı değer ile kendisine gelen değeri kıyaslar. Eğer iki değer aynı ise mesajın uygun olduğunu anlar ve alır, eğer farklı ise bunu bir hata durumu olarak algılar ve hata çerçevesi üretir. Bunu alan gönderici modül aynı mesajı tekrar göndermesi gerektiğini anlar ve gönderir.

Hattın sürekli meşgul olmaması için bir modül gönderdiği mesaj alındığında hattı bırakmalıdır. Bunu sağlayabilmek için geri bildirim biti kullanılır. Gönderici modül kendi çerçevesindeki acknowledge bitini her zaman çekinik tutar. Herhangi bir modül, bu mesajı beklesin veya beklemesin, mesajı düzgün olarak teslim aldığını göstermek için hatta dominant bit gönderir. Bu şekilde gönderici modül mesajın teslim edildiğini anlar ve hattı diğer modüllerin haberleşmesi için bırakır.

CAN BUS haberleşmesinde dış ortamdan ve etkilerden kaynaklı bozulmalar da olabilmektedir ancak bunlar için alınan önlemler ESD Devreleri bölümünde açıklanacaktır. Hatalar bulunduğunda bunlar giderilebiliyor da olsa sağlıklı bir haberleşme için CAN hattı üzerindeki modüllerin birbirlerine olan mesafeleri ve birbirleri ile olan uyumları mutlaka tasarım esnasında dikkate alınmalıdır.

1.10. ESD Koruma Devreleri

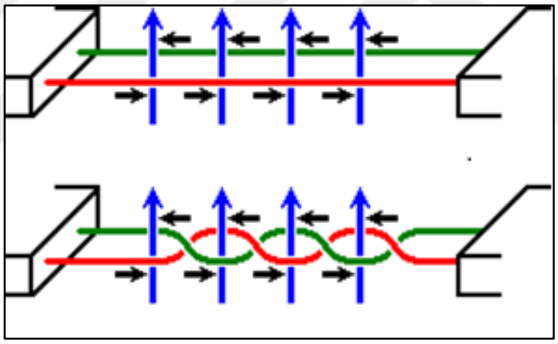
ESD devreleri oluşturulan hassas elektronik devreleri dışarıdan gelen çoğunlukla manyetik nedenlerden kaynaklı elektriksel etkilerden korumak amacıyla tasarlanırlar. Birbirine yakın duran birden fazla elektrikli komponent bulunan yapılarda sıklıkla karşılaşılan sorunların başında elektrostatik etkiler gelmektedir [30]. Bir elektrik devresi oluşturduğu manyetik alan dolayısıyla diğer devre üzerinde istenmeyen elektrik yükleri oluşmasına sebep olur ve o devreyi olumsuz etkiler.

CAN hatlarında haberleşme elektrik temellerine dayandığından dolayı hat üzerinde oluşabilecek istenmeyen bir yük, herhangi bir mesajın yanlış iletilmesine yada iletilememesine sebep olur ve hat üzerinde hatalar oluşur. Buna sebebiyet vermemek amacıyla CAN hatları özellikle yüksek akım taşıyan veya değişken elektrik yüklerinden dolayı manyetik alan oluşturabilecek elektrik devrelerinden ve anten tarzı

yapılardan mümkün olduğunca uzak tutulacak şekilde tasarım yapılır. Ancak yine de hatları korumak için bazı özel önlemler alınması gerekmektedir.

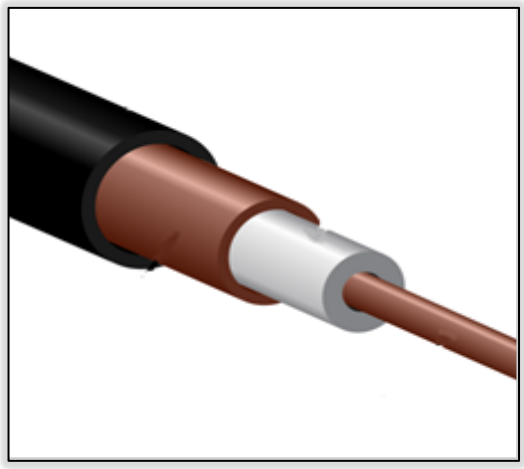
1.10.1. Burgulu ve koaksiyel kablo kullanımı

Burgulu kablo iki veya daha fazla sayıda kablonun birbiri üzerinden geçecek şekilde sarılması ile yapılan elektrik kablolarıdır. Manyetik etkilere hassas olacağı düşünülen sensor vb. yapılarda çoğunlukla ilgili besleme yada topraklama hattı ile data hattı twisted olarak kullanılır. Burgulu kablonun prensibi dışarıdan gelen manyetik etkinin kablonun daha küçük bir yüzey alanına temas etmesine ve etki ettiği muhtemel noktalarda oluşturduğu elektriksel akımın yönünün ters olmasına dayalıdır [31]. Bu şekilde elektriksel etkisi birbirini nötrlemekte veya minimuma inmektedir. CAN hatlarında neredeyse tüm uygulamalarda CAN H ve CAN L kabloları burgulu olarak kullanılmaktadır (Şekil 1.11).



Şekil 1.11. Burgulu kablo kullanımı [32]

Koaksiyel kablolar ise kablonun dışında bir metal kalkan olması prensibine dayanır (Şekil 1.12). Bu kalkan harici bir noktadan topraklanır ve faraday kafesi gibi davranır. Bu sayede dışarıdan gelen manyetik etkilere kalkan görevi görür [32]. Ekranlı kablolar daha güvenilir ve kesin çözüm olduğu halde pahalı olduğu için çok kritik uygulamalar dışında tercih edilmez. Otomotivde ve endüstride genellikle motorların çok yakınında bulunan hassas sensörler bu tip kablolarla haberleşirler.



Şekil 1.12. Koaksiyel kablo yapısı [32]

1.10.2. Kapasite kullanımı

CAN hatlarında kapasitif koruma ESD etkilerinden korunmak için bilinen en temel yöntemlerden biridir [33]. Fiziksel katmanı ilgilendiren bu yapılar anlık darbelerden hattı korumak anlamında etkilidir. Anlık ESD darbeleri çoğu zaman araçlar antenlerin yoğun olduğu bölgelerden geçtiği veya endüstriyel alandaki kullanımlarda manyetik alanın düzensiz olduğu bölgelerde olabilir.

Kapasitenin boyutunun büyük olması etkisini arttırır ve daha yüksek darbelere de dayanım sağlar ancak kapasite seçilirken fiziksel katmandaki sınırlamalara dikkat edilmelidir. Hattaki her modülün sahip olabileceği kapasitif yükün maksimum sınırı belirlidir ve bunun ötesine geçmemelidir. PCB üzerinde kapasitörlerin çıkış noktalarına olabildiğince yakın olması istenir. Bunun sebebi oluşan etkileri olası diğer noktalara sıçramadan elimine edebilmektir.

1.10.3. Sonlandırma dirençleri

Sonlandırma direnci CAN hattının iki ucunda bulunan çoğunlukla 120 ohm değerinde iki adet dirençten oluşur. Bu iki paralel direnç sayesinde CAN hattının toplam direnci 60 ohm civarlarında olmaktadır [34]. Haberleşme sistemlerinde bu yönteme DC sonlandırma adı verilir. Bununla birlikte bir kapasitör eklenmesiyle AC sonlandırma da yapılabilir. DC sonlandırma yapıldığında dirençlerin devreyi sürekli yüklemesi bir problem olarak görülmektedir. Bu sıkıntı gerilim farkına göre haberleşen sistemlerde gerilim farkının azalmasına sebep olur. AC sonlandırma yapıldığı taktirde bu problem

çözülür. Fakat kapasitör gerilimde gecikmeye sebep olmaktadır. Bu sebepten ötürü yüklenme hızının üzerinde çalışan şebekelerde hataya yol açabilir veya haberleşmeyi geciktirebilir.

Bu sebeplerden ötürü hızlı ve uzun kablo boyuna sahip haberleşme şebekelerinde DC sonlandırma tercih edilirken daha kısa ve haberleşme hızı olarak çok yüksek hızları kullanmayan şebekelerde AC sonlandırma yöntemi kullanılmaktadır. CAN hatlarında çoğunlukla DC sonlandırma tercih edilmektedir. Bu dirençler genellikle hat üzerindeki iki adet modüle entegre bulunurlar. CAN H ve CAN L hatları arası ölçüldüğünde 60 ohm civarlarında bir değer okunacaktır.

1.11. CAN Uyuma ve Uyanma Stratejileri

Haberleşme hatlarında bazı modüller sürekli olarak güce bağlı bazıları ise anahtarlanabilir güç kaynaklarına bağlıdır. Sürekli olarak güce bağlı olan modüller her an ciddi bir güç tüketmemeleri için uyku moduna geçebilme özelliğine sahiptir. Özellikle otomotiv dünyasında bu tip modüller araç kapalı iken aküyü tükettiğinden ötürü uyku modu oldukça önemli bir özellik olmaktadır. Uyku modunda bir modülün tükettiği akım 120 mA değerine veya daha aşağısına düşebilmektedir. Bu modüllerin sürekli güce bağlı olmasının sebebi belirli durumlarda aktif olmaları gerekmesidir. Örneğin arabalarda uzaktan kumanda ile kapıları kilitleyebilmek ve açabilmek için ilgili modülün sürekli olarak güce bağlı olması gerekmektedir aksi taktirde araba kapalı iken çalışmayacaktır.

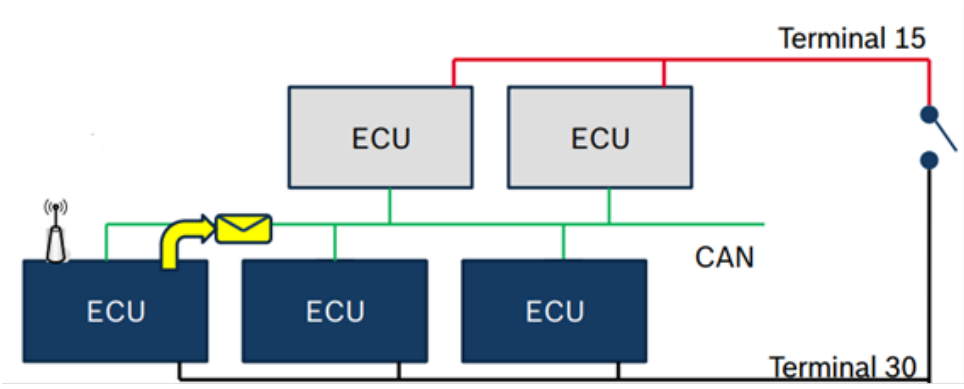
Bu tip modül yapıları tasarlanırken uyuma ve uyanma zamanlarına dikkat edilmelidir. Bir modül gerekmediği taktirde uyanık kalırsa çok fazla güç tüketecek, gereken zamanlarda uyanmazsa da fonksiyonel sıkıntılara sebebiyet verecektir [36]. Aynı zamanda, bir modül uyandığında ilgili diğer modülleri de uyandıracağı için ilgili tüm modüllerin de uyanma sebebine göre hareket etmesi ve diğer yapılara göre hata üretmemesi gerekir.

Bu tip modülleri dizayn edilirken mikroişlemci ve CAN kontrolörü biribirinden bağımsız olarak düşünülür. Mikroişlemci sürekli enerjili değildir ancak CAN kontrolörü sürekli olarak enerjilidir. Mikroişlemci güç alamadığı anda uyku moduna geçer ve beklemeye başlar. CAN kontrolörü veya herhangi başka bir yapı uyandığında

mikroişlemciye sinyal gönderir ve onu uyandırır. Belirli bir akımın üzerinde sinyal geldiğinde mikroişlemci uyanarak çalışmasını devam ettirir.

CAN hatlarında uyanma dominant bitler ile sağlanır. Hat uyku konumunda beklerken herhangi bir modül hatta dominant bit bastığında uyanacak şekilde dizayn edilmiş modüller bunu algılar ve mikroişlemcilerini uyandırır. Genellikle topolojilerde uyanma stratejileri CAN dışındaki yapılardan başlar ve CAN ile devam eder. Örneğin, arabanın kapılarını kilitleyip açan uzaktan kumanda mekanizmasını ele aldığımızda, modül içerisinde RF anteni kumandadan gelecek sinyali bekler ve sinyal geldiğinde uyanır. Devamında CAN hattı üzerinden ilgili diğer modülleri uyandırır ve kilitleri açmak ya da farları açmak gibi çeşitli davranışları gerçekleştirebilir (Şekil 1.13).

Modüller eğer Terminal 15 ile besleniyorlarsa çoğunlukla CAN ile uyanma söz konusu olmaz [36]. Terminal 15 otomotivde kontak anahtarı açıldığında gelen elektrik beslemesini temsil etmektedir. Ancak bazı modüller, örneğin kapı kilitlerini kontrol eden modül, Terminal 30'a bağlanmaktadır. Terminal 30 ise modülün direk aküye bağlı olması anlamına gelir. Bu durumda hem aküyü bitirmemek hem de kullanım senaryolarını yönetebilmek amacıyla CAN üzerinden uyuma ve uyanma tercih edilmektedir.



Şekil 1.13. CAN hattı RF ile uyanma örneği

Uyuma ve uyanma stratejileri ile alakalı dikkat edilmesi gereken nokta gecikme hataları oluşturmayacak şekilde bir topoloji tasarlamaktadır. Eğer herhangi bir modül uyanmaması gereken bir durumda uyanır ise beklediği bazı mesajları diğer modüller uyku modunda olduğu için alamayabilir. Bu durumda da bu mesajlar gelmediği için timeout hataları üretecektir. Bu modları yönetmek için bazı standartlar dizayn

edilmiştir ve endüstride çoğunlukla bu standartlar kullanılmaktadır. Bu sayede bilinen standart çalışma şekillerinde modüllerin uyuma ve uyanması için standart mesajlar basılmaktadır. Tüm modüller aynı standarda göre hareket ettiğinde uyuma ve uyanma stratejilerini yönetmek de kolay olmaktadır.

1.11.1. CAN topolojileri ve mesaj listeleri

CAN hattı sistemlerinin oluşturulmasından önce yapılması gereken ilk iş bir topoloji oluşturulmasıdır. Özellikle 10 veya daha fazla sayıda modül içeren CAN hatları çoğunlukla tek bir hattan oluşmaz. Hat yoğunluğunu ve mantıksal bütünlüğü yönetebilmek amacıyla aynı sistem içerisinde birden fazla CAN hattı kurulur ve araya konulan dönüştürücü modüller ile gerekli mesajların bir hattan diğerine transferi sağlanarak modüler bir yapı oluşturulur.

Topoloji, tüm CAN hatlarını ve bu hatlara bağlı olan modülleri gösterir. Bununla birlikte hangi modüllerin dönüştürücü modül görevi gördüğünü, hangi CAN hattının hangi hızda haberleştiğini, hangi fonksiyonel yapıların hat üzerinde bulunduğunu, modüllerin ID numaralarını ve hangi modüllerde sonlandırma dirençleri bulunduğunu da topolojilerde görmek mümkündür.

Topoloji oluştururken öncelikle fonksiyonel bütünlük göz önünde bulundurulur. Örneğin otomotiv topolojilerinde çoğunlukla motor ve aktarma organlarını ilgilendiren modüller bir CAN hattında, aracın gövde bölümündeki camlar ve kapı kilitleri gibi yapıları kontrol eden modüller farklı bir CAN hattında bulunurlar. Ancak bu yapıların biribiri ile de haberleşmesi gerekir. Kapıların araç belirli bir hıza ulaştığında otomatik olarak kilitlenmesi özelliğini örnek olarak alırsak, burada aracın hız bilgisini gönderen modül bir hatta kapı kilitlerini kontrol eden modül ise başka bir hatta bulunmaktadır. Bu durumda arada bulunan bir aracı modülün bu bilgiyi iletmesi gerekmektedir. Ancak hangi bilgilerin iletileceği doğru seçilmeli ve gerekli olmayan bilgiler hatlar arasında iletilmemelidir. Aksi taktirde hat yoğunlukları kullanılmayan mesajlardan dolayı çok yükselir ve hatları ayırmanın yoğunluğa olan faydası da ortadan kalkmış olur.

Hatların hızları ve güvenlik seviyesi de topoloji oluşturulmasında önemli rol oynamaktadır. Bazı CAN hatları modüllerin yazılımlarının yüklenmesi ve hata

ayıklama sistemlerinin kullanması amacıyla bulunmaktadır. Bu hatların daha yüksek hızlarda olması gerekmektedir. Aksi taktirde yazılım yükleme işlemleri çok uzun sürebilir. Hat hızı yükseldikçe ve bu tip yapılar işin içine girdikçe hattın güvenliği ve dayanıklılığı düşecektir. Bu sebepten ötürü kritik modüller ve yapılar bu hatlar yerine daha düşük hızlı CAN hatlarına koyulmaktadır.

Mesaj listeleri ise topolojide görünmeyen detayları gösterir. Mesaj ID leri ile birlikte mesajların içeriklerini, hangi modüllerin hangi mesajları gönderdiğini, hangi mesajların hangi hatlar arasında aktarıma dahil olduğunu ve hepsinden önemlisi hangi mesajın hangi bayt ve bitinin hangi anlamı ifade ettiğini mesaj listesinde görmek mümkündür. Hat üzerinde bulunan bütün modülleri konuşurken mesaj listesi ile birlikte oluşturulan bir dili konuşurlar. Bu sayede bir mesajı gönderen modül ve dinleyen diğer tüm modüller için aynı bit ve baytlar aynı anlamları ifade eder.

Mesaj listeleri aynı zamanda mesajların gönderilme sürelerini de belirler. Bazı mesajlar saniyede bir kere gönderilirken bazıları saniyede 100 defaya kadar gönderilebilir. Bu tamamiyle mesajın taşıdığı bilginin hangi işler için kullanılacağı ve bu işlerin hangi öncelikte ve hangi tepki süresinde yapılması gerektiği ile alakalıdır.

1.11.2. CAN FD yapısı

Otomotiv endüstrisi başta olmak üzere CAN haberleşmesinin kullanıldığı birçok alanda haberleşme hattına bağlanan modül sayıları ve haberleşme yoğunluğu seneler içerisinde artış göstermektedir. Daha fazla modülün haberleşme yapısına dahil olması ile birlikte CAN haberleşmesinin mevcut hızı ve kapasitesinin yetersiz kaldığı görülmektedir. Bu sebepten ötürü CAN FD yapısı 2011 senesinde geliştirilmiş ve Bosch tarafından 2012 senesinde yayınlanmıştır [37].

Standart CAN haberleşmesinde ulaşılabilecek maksimum hız 1 mbps olarak bilinmektedir. CAN FD ile bu hızın 5 mbps seviyelerine çıkabildiği görülmektedir. Bu şekilde paylaşılan data gerçek zamanlı data elde etmeye bir adım daha yaklaşmaktadır. CAN FD yapısının otomotiv sektörü ile birlikte Endüstri alanında da yaygın olarak kullanılması beklenmektedir.

1.12. Endüstri 4.0

Endüstri 4.0 yada başka bir deyişle dördüncü sanayi devrimi; 2020'li yıllarda tam anlamıyla hayatımıza girmesi beklenen, üretim başta olmak üzere endüstrinin tüm alanlarında Nesnelerin İnterneti ve siber fiziksel sistemlere dayalı yapıların kullanılmaya başlamasını kapsar [38]. Hayatımızın her alanında Nesnelerin İnterneti kavramının her geçen gün daha da fazla yer aldığını görüyoruz. Hayatı kolaylaştıran ve etrafımızdaki tüm nesneleri akıllı hale getirmeyi hedefleyen bu hareketin Endüstri alanına da sıçraması kaçınılmazdır. Endüstri 4.0 içerisinde Nesnelerin İnterneti uygulamalarının ve özellikle Büyük Veri işleme ve anlamlandırma yapılarının hem üretim süreçlerini hızlandırması hem de toplanan verilerin daha hızlı ve doğru değerlendirilmesi yoluyla aksaklıkları daha hızlı saptayarak yapılacak iyileştirmelerde ve süreç yönetiminde büyük kolaylıklar sağlaması beklenmektedir [39].

Sanayi Devrimlerinin geçmişine baktığımızda ilk sanayi devriminin buharlı iş makinalarının bulunması ile 1763 yılında başladığını görmekteyiz. Bu ilkin ardından endüstri hızla gelişimini sürdürmüş ve elektrikli üretim aletlerinin işin içine girmesi ile birlikte 1870 yılında ikinci sanayi devrimi gerçekleşmiştir. Bunun devamında üretimin insan eliyle yapıldığında ortaya çıkan hatalardan nasıl arındırılabileceği tartışılmaya başlanmıştır. 1969 senesinde ilk PLC (Programlanabilir Lojik Kontrolör) yapısının bulunması ve kullanılması ile birlikte üçüncü sanayi devrimi gerçekleşmiştir. Bu gelişme ile birlikte ilk defa otomasyon kavramı endüstride konuşulmaya başlanmış ve insan eli değmeden üretim yapılmaya başlanmıştır [40]. Yine insanlar tarafından programlanan makinaların söz konusu işleri yapmasının ardından bir sonraki adım bu makinaların birbirleri ile iletişimi olarak görülmüş ve Nesnelerin İnterneti ile yayılan bağlı cihazlar kavramının endüstride kullanımı gündeme gelmiştir. Endüstri 4.0 bu yapıların devreye girmesi ile birlikte bu aşamanın da başarılı olmasını hedeflemektedir. Bu noktada fabrikaların tamamen makina gücü ve akıllı sistemler ile üretime geçmesi ve üretimi hızlandırmanın yanında hatalar ile gelen maliyetleri de minimuma indirmesi hedeflenmektedir.

Üretim sektöründe sistemin dayanıklılığı diğer tüm parametrelerden önce gelmektedir [41]. Herhangi bir arıza olduğunda üretim hattının durması çok kısa zamanlarda bile ciddi maddi kayıpları beraberinde getirmektedir. Bu sebepten ötürü, özellikle üretim

alanında kullanılan yapılarda arıza oranının çok düşük ve arıza olduğu durumlarda giderilmesinin olabildiğince hızlı olması beklenmektedir.

CAN haberleşmesinin özellikle güvenlik anlamında getirdiği avantajlardan dolayı Endüstri 4.0 ile birlikte kullanımının ciddi oranda artması beklenmektedir. Sistemin yönetildiği yapılar özellikle uzaktan güncelleme alabilmeleri ve raporlama yapabilmeleri için internete bağlı olmak zorunda olsa da sistemi oluşturan tüm alt elemanların özellikle üretim hatları gibi çok uzun mesafeli olmayan durumlarda CAN ile haberleşmesi hedeflenmektedir [42].

Günümüzde yaygın olan CAN haberleşme yapıları çoğunlukla 500 kbps hıza kadar çıkabilmektedir ve bu üretimle ilgili endüstriyel uygulamalarda yetersiz kalabilmektedir. Bu sebepten ötürü şu anda 1 mbps üzerinde hızlara çıkabilen CAN FD yapısı çalışılmakta ve kullanımı gün geçtikte artmaktadır. Endüstri 4.0 uygulamalarında da en yaygın kullanılacak yapının CAN FD olması beklenmektedir.

Bu yapının ilk örnekleri yangın alarm sistemlerinde kullanılmaya başlanmıştır. Binaların yangın alarm sistemleri anlık olarak tepki verebilmesi ve çevredeki binalara olan riskleri de bildirebilmesi için tek bir ağ üzerinden haberleşebilmektedir. Binaların hepsi birbiri ile ve yangın anında müdahale için bir veya birden fazla nokta ile haberleşirken binaların içerisindeki yangın sensörleri söndürmeye yardımcı sistemler bu haberleşmeye internet üzerinden dahil olmamaktadır. Bu yapıların bir tanesinin arızalanması veya hatalı veri göndermesi durumu ciddi etkiler doğurabileceğinden dolayı tüm sensör ve müdahale sistemleri binaların kendi içerisinde CAN ile haberleşmektedir. Özellikle sensörlerin konumlanması buna göre planlanmakta ve bir tanesi arıza bile yapsa diğerleri çalışmasını aynı şekilde sürdürebildiği için binadaki bir yangının sensörlerden bir yada birkaçı arızalansa bile hata olmaksızın algılanabilmesi sağlanmaktadır.

2. MALZEME VE YÖNTEM

2.1. Eğitim Seti Çalışması

2.1.1. Amaç ve kapsam

Bu çalışma kapsamında CAN haberleşmesinin yukarıda belirtilen özelliklerini ortaya çıkartmak amacıyla bir eğitim seti tasarlanması amaçlanmıştır. Eğitim setinin öncelikli amacı çalışan bir CAN Bus haberleşme hattı kurmak, sonrasında ise mesajların basılma durumlarını kontrol ederek olası hata durumlarını ve CAN hattının bu hatalara verdiği tepkileri gözlemleyebilmektir.

Bu set yardımıyla 3 veya daha fazla modülden oluşan bir CAN hattının olası davranışları gözlemlenebilecek, kontrol edilebilecek ve standart bir CAN topolojisinin nasıl oluşturulduğu çalışılabilecektir.

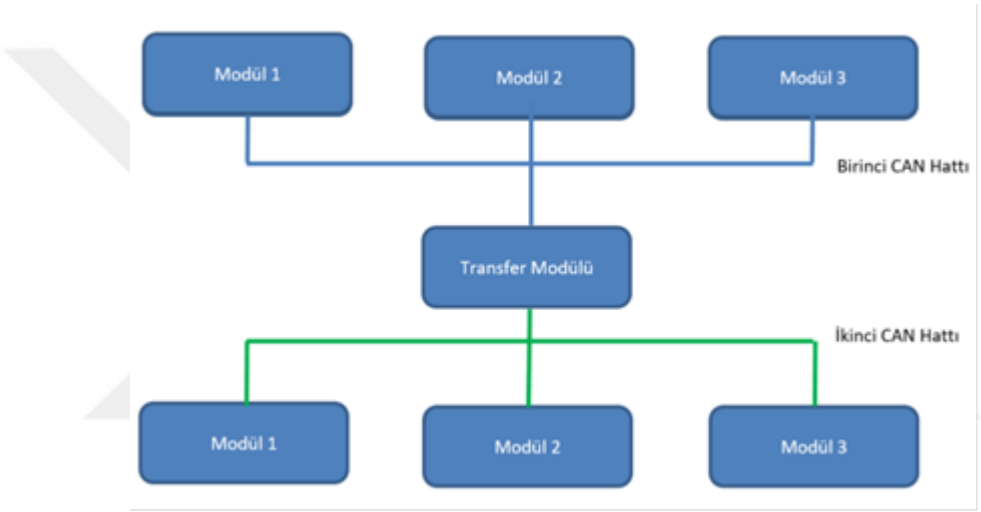
Gerçek zamanlı çalışmanın yanı sıra set içerisindeki simülasyonlar ve bilgi ekranları sayesinde CAN hattının çalışma şeklini örneklerle anlama ve öğrenme imkanı sunulacaktır.

2.1.2. Örnek alınan senaryo

Endüstride CAN haberleşmesinin en yaygın ve efektif kullanımına otomotiv sektöründe rastlanmaktadır. Bu sebeple eğitim setinin tasarımında otomotivde kullanılan CAN haberleşme sistemi örnek alınmıştır. Otomotiv endüstrisinde neredeyse her araçta bulunan standart modüller referans alınmış ve kullanılmış olup, bu sayede eğitim seti kullanacak kişileri sektörde karşılaşacakları olası senaryolara da hazırlamayı hedeflemektedir.

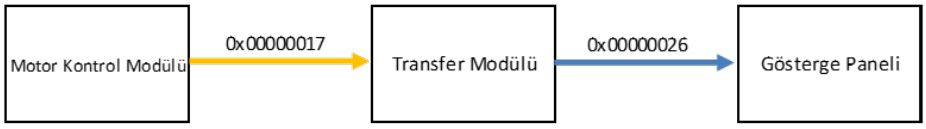
Otomotivde çoğunlukla tek bir CAN hattı bulunmaz. Otomotiv CAN topolojileri çoğunlukla mantıksal bütünlüğe göre ayrılan birden fazla CAN haberleşme hattından meydana gelir. Bu hatların çoğu fonksiyonun çalışması için birbirlerine de bağlı olması

ve mesaj alışverişi yapması gereklidir. Örneğin, gösterge paneli ve takograf gibi sürücü bilgi sistemleri için bir hat varken motor ve kalibrasyon sistemleri için ayrı bir motor CAN hattı mevcut olabilir. Ancak gösterge panelinin motor arıza durumunu veya aracın motor devrini gösterebilmesi için bu iki CAN hattının biribirine erişebilmesi ve gerekli mesajları alabilmesi gereklidir. Bu tip durumlar için transfer(gateway) modulü adı verilen modüller kullanılır [43]. Transfer modülleri birden fazla CAN hattına bağlanır ve istenilen mesajları hatları arasında transfer eder. Bu sayede bir hatta basılan bir mesaj diğer tüm hatlarda ihtiyaç olunması halinde bulunabilir ve ilgili modüller tarafından kullanılabilir (Şekil 2.1).



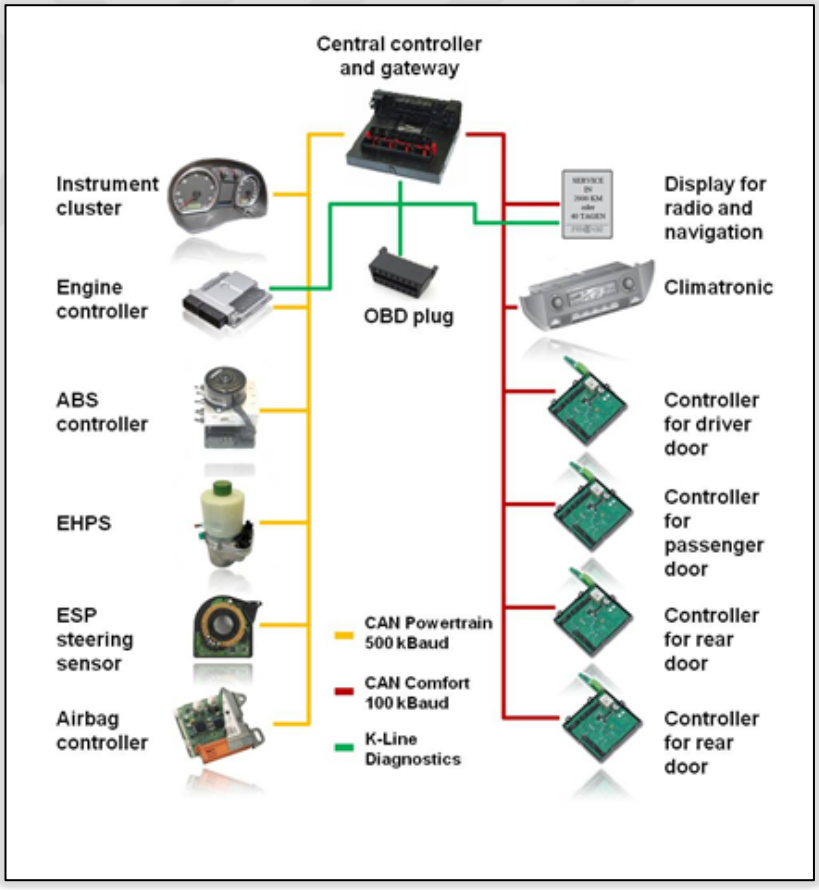
Şekil 2.1. CAN transfer modülü topoloji örneği

Bu yapıda dikkat edilmesi gereken en önemli nokta, transfer işleminden sonra CAN mesaj ID numarasının aynı kalmayacağıdır. Mesaj transfer modülünden geçtikten sonra o modülün ID numarası ile diğer hatta basılacaktır. Bu sebeple topolojiler ve mesaj listeleri oluştururken çoğunlukla transferden geçecek mesajlar için ayrı sekmeler oluşturularak takip edilir. Örnek resimde görüldüğü üzere aynı mesaj ilk başta 17 modül ID numarası ile transfer modülüne gönderilmekte, devamında ise 26 modül ID numarası ile gösterge paneline iletilmektedir (Şekil 2.2).



Şekil 2.2. Mesaj transferi sırasında ID değişimi

Bir başka önemli nokta ise gecikme konusudur. CAN hatları biribirinden ayrılırken mantıksal bütünlüğe göre ayrılmasının tercih ediliyor olmasında gecikme faktörü en önemli etmendir. Biribiri ile sürekli iletişim halinde olan ve çalışan modüller arasındaki kritik mesajlarda gecikme olmamalıdır. Araya bir transfer modülü girdiğinde ufak da olsa bir gecikme mutlaka yaşanmaktadır. Şanzıman sistemi veya fren kontrolü gibi konularda bu tip gecikmeler tölere edilemez sonuçlara yol açabileceğinden dolayı bu sistemler ile ilgili birlikte çalışan modüller çoğunlukla aynı hat üzerinde yer almaktadırlar. Gösterge paneli veya araç içi eğlence sistemi gibi yapılarda ise gecikmeler tölere edilebilirdir. CAN hattında transfer dolayısıyla yaşanabilecek gecikmeler gözle görülecek etkiler yaratmadığı için bu tarz sistemlerde öncelikli olarak değerlendirilmemektedir.



Şekil 2.3. Transfer modülü diagnostik bağlantısı [44]

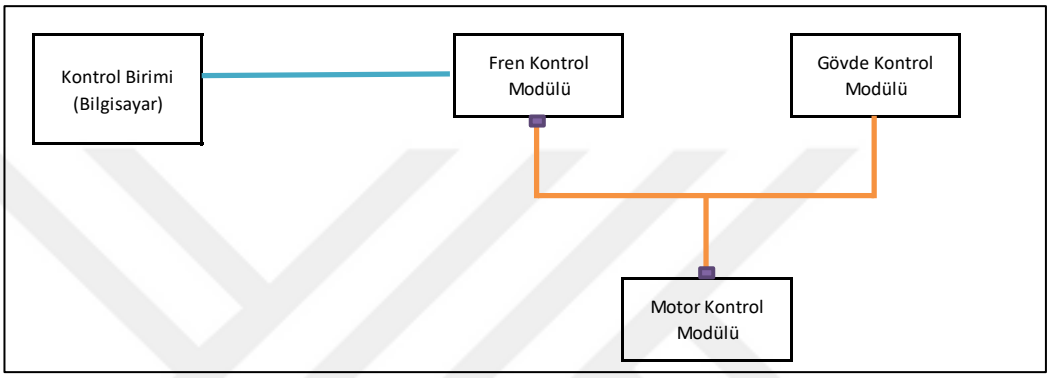
Transfer modüllerinin bir başka avantajı da diagnostik amacıyla kullanılabilmeleridir [45]. Birden fazla CAN hattına bağlı oldukları ve bu hatlardaki tüm modüllere erişimleri olduğu için bu modüller çoğunlukla diagnostik ve yazılım atma amaçlarıyla

da değerlendirilmektedir. Resimde görüldüğü gibi birden fazla hatta bulunan modüller tek bir transfer modülüne bağlanmakta ve sonrasında OBD konnektörü [46] yardımıyla diagnostik amacıyla kullanılabilecek hale gelmektedir. Eğitim setinde de buna benzer bir amaçla kullanılacak bir transfer modülü bulunacak ve bilgisayar sistemine bu modül ile bağlanılacaktır (Şekil 2.3).

3. BULGULAR VE TARTIŞMA

3.1. Topoloji ve Açıklama

Eğitim seti topolojisi Motor Kontrol Modülü, Gövde Kontrol Modülü ve Fren Kontrol Modülü olmak üzere üç adet araçta yer alan modül, bir adet ana transfer modülü ve bilgisayar kontrol arayüzünden oluşmaktadır. Eğitim seti topolojisi şematik olarak Şekil 3.1'de gösterilmiştir.



Şekil 3.1. Eğitim seti topolojisi

Tüm modüllerin otomotiv standartlarında olduğu gibi kendilerine düşen görevi yerine getirecek şekilde çalışması planlanmıştır. Bu şekilde setin gerçek endüstri standartlarına uygun olması ve bu anlamda da bir kazanım elde edilmesini sağlamasıdır.

Motor kontrol modülü motor kontrolü ile ilgili parametreleri barındırır ve diğer modüller için gerekli bilgileri araç CAN hattına gönderir. Bu bilgilerden set içerisinde kullanılacak olanlar araç hızı, motor devri ve vites bilgisidir. Vites bilgisi otomatik vites araçlarda şanzıman modülleri tarafından işlenir ve gönderilir ancak manuel vites arabalarda motor kontrol modülü tarafından algılanmakta ve CAN hattına basılmaktadır. Set içerisinde modül sayısını istenilen durumda tutabilmek adına manuel araç referans alınmış ve vites bilgisi motor kontrol modülü içerisinde bir parametre olarak değerlendirilmiştir.

Gövde kontrol modülü, araç gövdesinde daha çok kullanıcının doğrudan ilişkisi olan fonksiyonları kontrol etmekte ve yönlendirmektedir. Bu fonksiyonlar ile ilgili bilgiler de doğrudan bu modül aracılığı ile araç CAN hattına gönderilmektedir. Eğitim seti içerisinde, gövde kontrol modülüne dair bir parametre kontrol edilmeyecektir ancak

bu modül CAN hattını okuma ve değerlendirme yapabilmesi amacıyla yine de sette tutulmuştur. Otomotiv örneğinde önemli bir yere sahip olması da bu modülü bulundurma amaçları arasında gösterilebilir.

Fren kontrol modülü (EBS/ABS modülü) araçlardaki tüm fren sistemini ve ABS/ASR gibi yardımcı sistemleri kontrol eden modüldür. Bu modül araç üzerinde fren bilgisi dışında ivmelenme bilgisi ve fren balata sıcaklığı gibi önemli bazı değerleri de araç CAN hattına göndermektedir. Bu eğitim seti kapsamında frene basılma durumunu gösteren modül olarak kullanılmaktadır. Bununla birlikte fren kontrol modülü transfer modülü görevi de görmektedir. Frene basılma bitinin değişimine göre set hangi senaryo durumunda olduğunu anlamakta ve buna göre tepki vermektedir.

3.2. Set Özellikleri

Eğitim seti hem yazılı olarak hem de simülasyonlar ile CAN haberleşmesinin ön plana çıkan özelliklerini göstermek ve deneyimlenmesini sağlamak üzere tasarlanmıştır. İçerisindeki modüller ile CAN haberleşmesi ile alakalı konular en temelden başlayacak şekilde incelenip anlaşılabilir ve sonrasında simülasyonlar ile görsel olarak da tecrübe edilebilir.

Eğitim modülü içerisindeki konu başlıkları şu şekilde sıralanabilir:

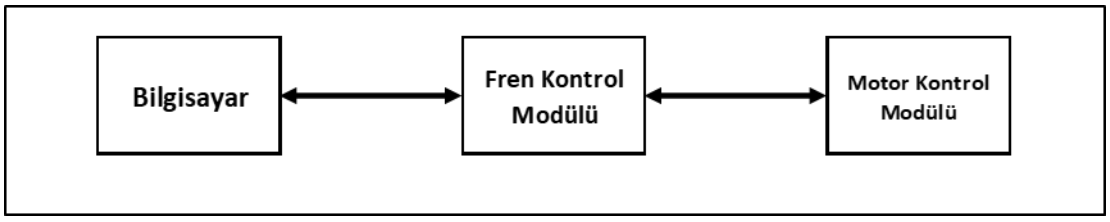
- CAN Tarihçesi
- CAN Mimarisi
- Avantajları ve Kullanım Sebepleri
- Bilinen CAN Protokolleri
- Veri Yolu ve Bağlantı Noktaları
- Mesaj Önceliklendirmesi
- Multimaster Çalışma Yapısı
- CAN Mesaj Yapıları
- Kuanta Kavramı ve Bit Yapısı
- Örnekleme Zamanı
- Hat Yoğunluğu
- Hata Çerçeveleri
- Zaman Gecikmesi Hataları

- Hata Önleme Yapıları
- TEC ve REC Registerları
- Hakkında
- Genel Bilgiler
- Web Sitesi linkine yönlendirme
- Simülasyon ekranına yönlendirme

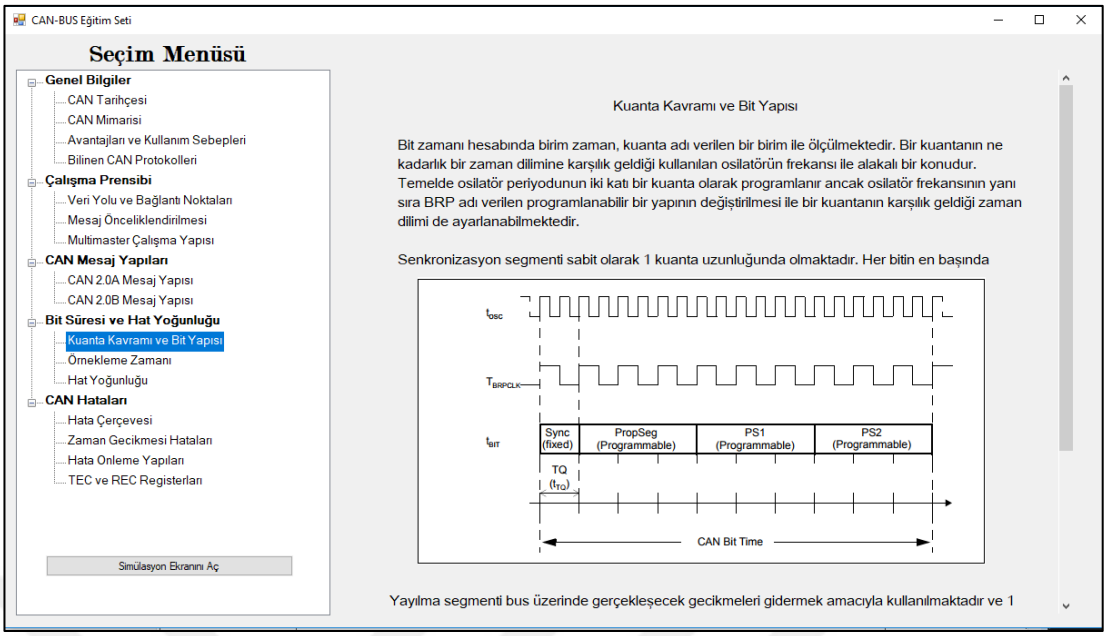
3.3. Set Çalışma Prensibi

Topolojiden de anlaşılacağı üzere setin çalışma prensibi bilgisayar üzerinden gönderilen taleplere bir adet transfer modülünün cevap vermesi ve diğer CAN hattına bağlı modülleri ilgili şekilde yönlendirmesi üzerinedir. Bu noktada transfer modülü ile bağlı bilgisayar seri port yardımı ile haberleştirilecektir. Bilgisayar arayüzü .NET platformu üzerinde Visual Studio ortamında geliştirilmiştir. Burada kullanıcının talepleri yine seri port haberleşmesi yardımıyla transfer modülüne gönderilmektedir.

Transfer modülü ile diğer modüller arasındaki haberleşme CAN ile sağlanmaktadır. CAN mesajları içerisindeki belirli bitler bu amaçla ayrılmış olup sadece transfer görevi gören modül ile diğer modüller arasında bilgisayardan gelen komutların paylaşılması amacıyla kullanılmaktadır. Eğitim seti çalışma prensibi Şekil 3.2'de şematik olarak gösterilmiştir. Eğitim seti arayüz görüntüsü Şekil 3.3'de setin görseli ise Şekil 3.4'de sunulmuştur.



Şekil 3.2. Eğitim seti çalışma prensibi örneği



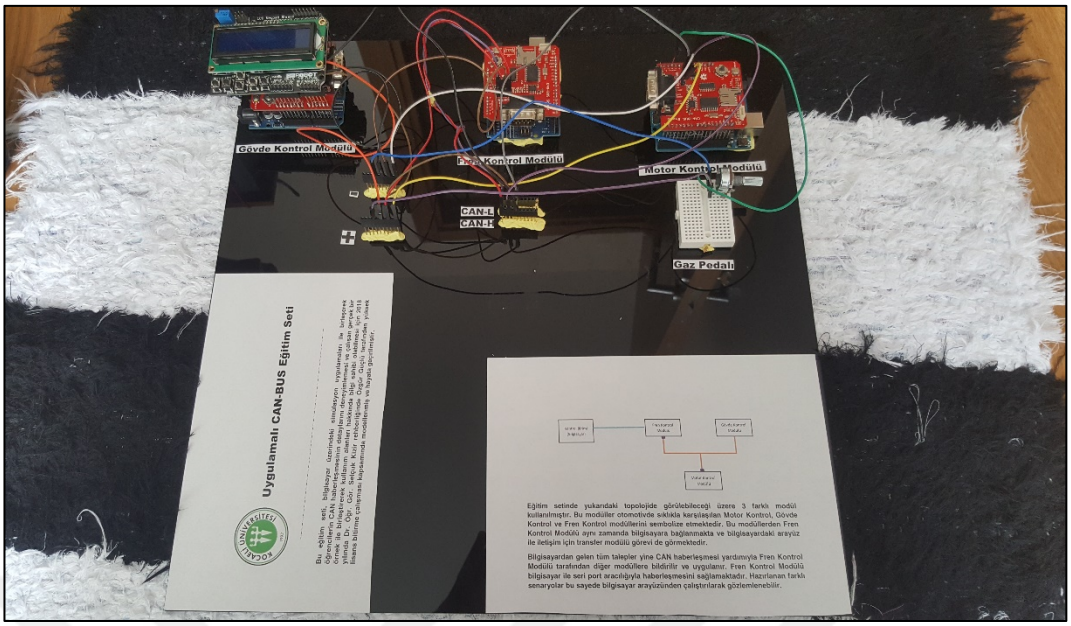
Şekil 3.3. Eğitim seti arayüz görüntüsü

Sette yer alan modüller için belirlenen modül ID leri Tablo 3.1'de görülebilmektedir. Bütün mesajlaşma bu 4 ID arasında geçecektir. Setin endüstrideki eşdeğerlerini temsil etmesi için modül ID numaraları otomotiv standartlarına uygun olarak seçilmiştir.

Tablo 3. 1. Eğitim seti modül ID bilgileri

Modül ID Bilgileri	
Modül Adı	**Modül ID**
Gövde Kontrol Modülü	0x21
Motor Kontrol Modülü	0x11
EBS / Fren Kontrol Modülü	0x22

Transfer görevi yapan modülden giden CAN mesajının en son baytı diğer modüllerle iletişim amacıyla ayrılmıştır. Bu bayt içerisinde yer alan 8 bitlik bilgi bölümü bilgisayar arayüzünden gelen talepleri taşır ve diğer modüllere hangi senaryoyu çalıştıracaklarını haber verir. Gerçek zamanlı simülasyon kısmından istenilen senaryo seçilerek bilgisayarın modüllere haber göndermesi sağlanabilmektedir.



Şekil 3.4. Eğitim seti görseli

Bu set kapsamında 4 senaryo ele alınmıştır. Bunlardan ilki normal çalışma senaryosudur. Hiç bir işlem yapılmadığı durumda modüllere enerji verildiğinde bu durumda olacaklardır. Bu durumda modüller mesaj listesinde bulunan tüm mesajları değiştirmeden standart değerlerde gönderirler. Araç hızı 100 km/saat yada motor devrinin 3500 rpm olması gibi düşünülebilir.

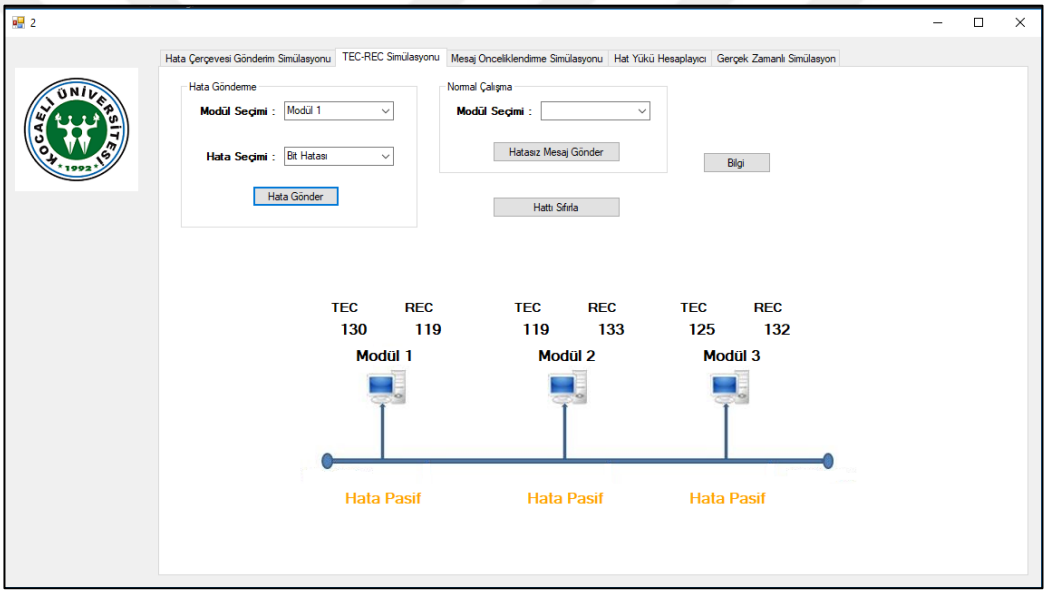
İkinci senaryoda ise aracın fren yaptığı durum ele alınmıştır. Fren yapma durumunda fren kontrol modülü bunu algılar ve frene basıldığını bildirir. Bu durumda aracın hızının ve motor devrinin de azalması beklenir. Bu senaryo çalıştırıldığında CAN hattına basılan mesajlar değiştirilerek araç hızının ve motor devrinin azalması, frene basılma durumunun ise güncellenmesi gözlemlenecektir.

Üçüncü senaryoda aracın vites arttırma durumu göz önünde bulundurulmuştur. Bu noktada araç hızına etki edilmemiş ancak aracın vites bilgisi güncellenmiş ve buna bağlı olarak motor devri değişimi gösterilmiştir. Vites arttırma durumu söz konusu olduğundan motor devrinde de anlık bir azalma gözlemlenecektir.

Dördüncü ve son senaryoda ise aracın vites düşürme durumu göz önünde bulundurulmuştur. Üçüncü senaryonun aksine burada aracın motor devrinin artması beklenmektedir. Aynı zamanda aracın bulunduğu vites konumu da güncellenecektir. Bu senaryoda da aynı üçüncü senaryoda olduğu gibi araç hızına bir etki gözlemlenmeyecektir.

Bu senaryolar ile oluşturulan durumların kolaylıkla takip edilebilmesi için set üzerindeki LCD ekrandan faydalanmak mümkündür. Daha rahat anlaşılabilmesi için normal çalışma senaryosu dışındaki tüm senaryolarda CAN mesajlarının güncellenme hızı yarı yarıya düşürülmüştür.

Gerçek zamanlı simülasyon dışında eğitim seti kapsamında CAN haberleşmesini anlatan simülasyonlar da çalışılmıştır. Bu simülasyonlarda hata çerçevesi gönderimi, TEC ve REC register değerlerinin değişimi, mesaj önceliklendirmesinin gösterimi ve eklenen mesajlara göre hat yükünü hesaplayan bir algoritma eklenmiştir. TEC ve REC registelarını gösteren simülasyonun örnek ekranı Şekil 3.5'te sunulmuştur. Eğitim setinin görseli ise Şekil 3.6'da verilmiştir.



Şekil 3.5. TEC-REC Simülasyon Ekranı

3.4. Mesaj Listesi ve Açıklamaları

Mesaj listesi içerisinde belirlenen mesajlar setin diğer bölümlerinde olduğu gibi endüstriyel otomotiv standartlarından alınmış ve eklenmiştir. Mesaj listesi içerisinde belirli sinyallerin ortak mesajların içerisinde oldğu görülmektedir. Bunun mantıksal sebepleri bu bölümde açıklanacaktır. Mesajların basılma hızları ise standart olarak saniyede 1 defa olarak belirlenmiş ve set içerisinde girilmiştir. Aşağıdaki tablolarda üç adet modülün hepsi için ilgili mesajlar ve detayları verilmiştir.

Fren kontrol modülü için bu çalışma kapsamında incelenen mesajlar özelinde böyle bir ayrıma gerek olmadığı görülmüş ve tek bir mesaja indirgenmiştir. Fren kontrol

modülü kodları Ek-1a'da verilmiştir. Motor kontrol modülü motor ve vites bilgilerini göndermektedir. İçerisindeki kodlar Ek-1b'de verilmiştir. Gövde kontrol modülü burada okuma görevi görmektedir. İçerisindeki kodlar Ek-1c'de verilmiştir. Modüllere ait mesajların detaylarını gösteren mesaj listeleri Tablo 3.3 ve Tablo 3.4'de verilmiştir.

Bilgisayar üzerindeki yazılım simülasyonlar ile birlikte tüm bu modüllere senaryolara göre gereken mesajları da göndermekte ve tüm sistemi kontrol etmektedir. Visual Studio ortamında C# dilinde geliştirilen bu yazılımın kodları Ek-2a ve Ek-2b'de bulunabilir.

Tablo 3. 2. Motor kontrol modülü mesaj listesi

Modül ID	Mesaj ID	Mesaj Adı	Sinyal Adı	Boyut	Bayt	Bit Pozisyonu	Detaylı Açıklama	Birim	Çözünürlük	Durum
0x11		Araç Hız Mesajı	Araç Hızı	8	1	8~1		km/h	1/256	
		Motor Bilgi Mesajı	Araç Motor Devri	8	1	8~1		rpm/bit	0.125	
		Vites Bilgi Mesajı	Vites Durumu	2	1	2~1		SED	1	
							Vites Konumu : 4			0x0
							Vites Konumu : 3			0x1
							Vites Konumu : 5			0x2
							Geçersiz			0x3

Tablo 3. 3. Fren kontrol modülü mesaj listesi

Modül ID	Mesaj ID	Mesaj Adı	Sinyal Adı	Boyut	Bayt	Bit Pozisyonu	Detaylı Açıklama	Birim	Çözünürlük	Durum
0x22		Fren Bilgi Mesajı	Fren Durumu	2	1	2~1		SED	1	
							Normal Çalışma Durumu			0x0
							Frene Basıldı			0x1
							Senaryo 3			0x2
							Senaryo 4			0x3

Mesaj listelerinde görüldüğü üzere modül ID numaraları belirtilmiştir. Devamında mesajın ve altında bulunan sinyallerin isimleri verilmiştir. Bu isimler sadece kullanımı ve anlamayı kolaylaştırmak içindir. Herhangi bir standarda bağlı değildir ve değiştirilebilirler.

İlgili sinyalin boyutu ve bayt, bit pozisyonları önemlidir. Mesajın 8 baytlık biriminin hangi bölümünde hangi sinyalin yer aldığı burada anlaşılır ve alıcı modüller mesajları bu noktalara göre filtrelerler.

Normal şartlarda bir bitlik bölümde gönderilebilecek, iki farklı değere sahip mesajların 2 bitlik paketler ile gönderildiği gözükmektedir. Bu ayarlama modüllerin hata yakalama ve işleme algoritmaları ile yakından alakalıdır. Hata ve mevcut değil durumları eğer gönderici modül tarafından algılanır ise hatta önceden tanımlı bir mesaj olarak basılması CAN hattına zarar vermesini ve hata oluşturmasını önleyeceği gibi

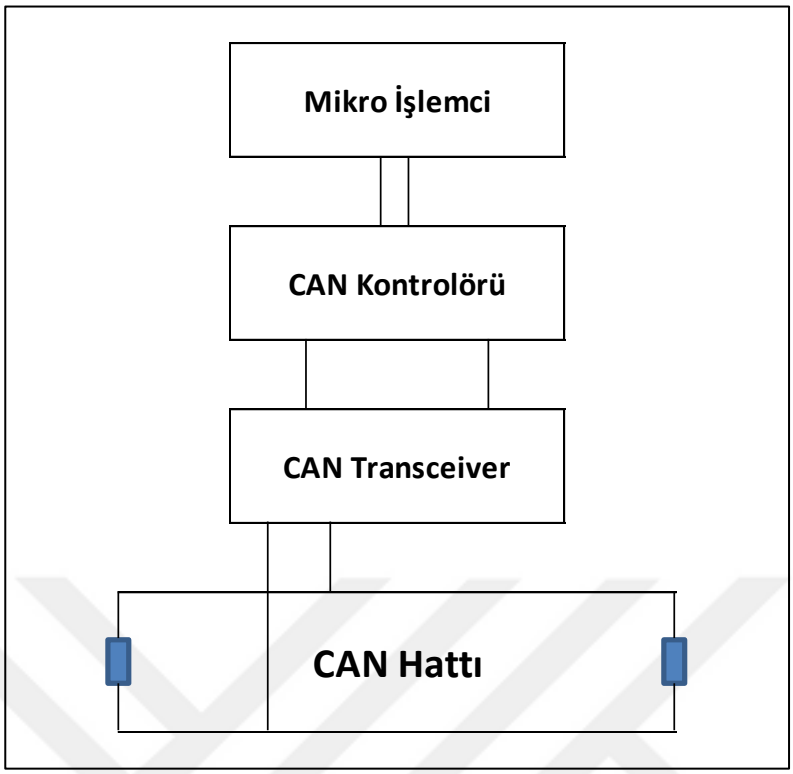
alıcı modülün de buna göre hazırlık yapmasını ve gerekiyorsa bazı fonksiyonlarını korumaya almasını sağlayabilir. Hata durumu çoğunlukla modüller beklemediği seviyede bir sensör sinyali ile karşılaştığında ortaya çıkar. Bazı nadir durumlarda ve kritik parametrelerde ise modüller iki farklı kaynaktan aynı bilgiyi talep ederek doğrulama algoritmaları çalıştırırlar. Bu kaynaklardan herhangi biri farklı sinyal gönderdiğinde modül bunu hata olarak algılayabilir. Mevcut değil durumu ise bağlantılarda bir sıkıntı olduğunda veya gönderici modül herhangi bir sebepten dolayı o değeri hesaplayamadığında oluşabilir. Genellikle opsiyonel sunulan özelliklerde bazı mesajlar kapatılır. Bu gibi durumlarda mevcut değil mesajı basılarak diğer tüm modüllerin o fonksiyonun çalışmayacağını anlaması sağlanmaktadır.

3.5. Kullanılan Parçalar ve Devre Şemaları

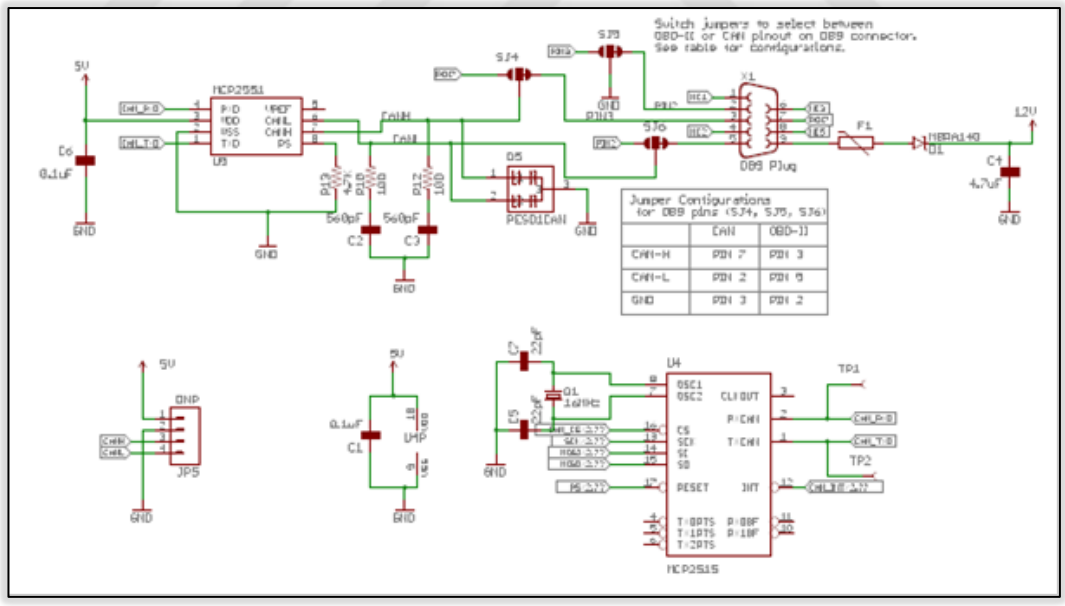
Setin yapımında modül işlemcileri için Arduino modülleri kullanılmıştır. CAN hattındaki modüller için Atmega 328 işlemcisi kullanan Arduino UNO kullanılmıştır.

CAN haberleşmesi için ise kullanılan işlemciler kendi CAN kontrolörlerini içermediğinden dolayı CAN kontrolörü ve CAN alıcı-verici devresi ayrı parçalar olarak kullanılmıştır. CAN kontrolörü olarak MCP 2515 entegresi [47], CAN alıcı-verici görevini üstlenmesi için ise MCP 2551 entegresi [48] tercih edilmiştir. Sıklıkla kullanılan entegreler oldukları için bulunmaları ve gerektiği durumda değiştirilmelerinin daha kolay olacağı planlanmıştır. Şekil 3.6'da CAN fiziksel katman yapısı gösterilmiştir.

MCP 2515 ve MCP 2551 entegreleri için kullanılan devre üzerindeki bağlantı şemaları aşağıda Şekil 3.7'de verilmiştir. İşlemci ve uyumluluğu ve modüler yapısı nedeniyle bu devreler tercih edilmiştir.

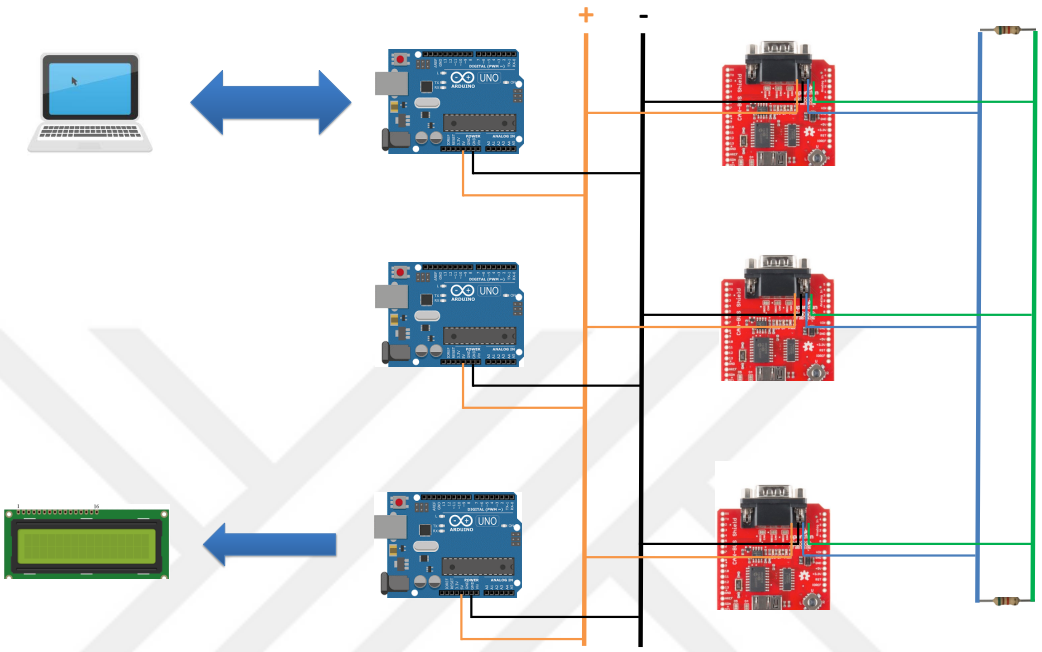


Şekil 3.6. CAN fiziksel katman yapısı [49]



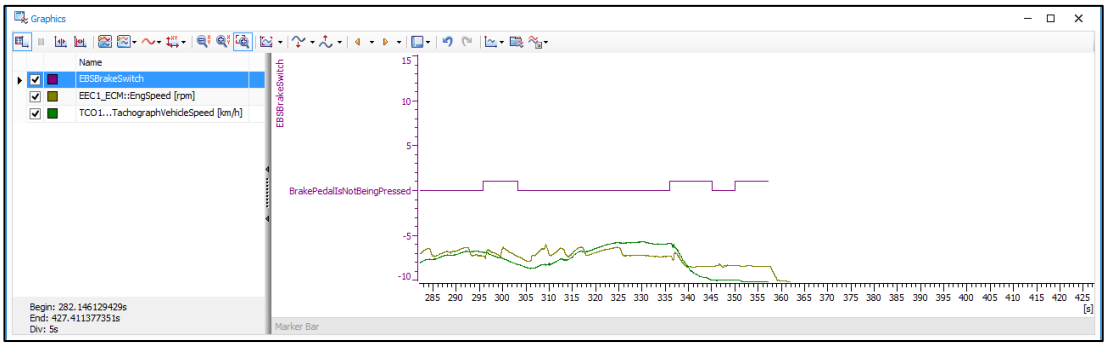
Şekil 3.7. MCP2551 ve MCP 2515 bağlantı şemaları [50]

CAN haberleşmesi için kullanılan eklenti devreleri ile bir CAN hattı oluşturulmuş ve sonlandırma dirençleri ile desteklenmiştir. Setin CAN haberleşmesi için kurulan devre şeması aşağıda Şekil 3.8'de görülmektedir. Set üzerinde bulguların görülebilmesi için Gövde Kontrol Modülü'ne bağlı bir LCD ekran bulunmaktadır.



Şekil 3.8. CAN haberleşmesi için kurulan devre şeması

Haberleşme sisteminin çalışır hale gelmesinin ardından CAN haberleşmesi otomotiv sektöründe araçlarda hata bulmak ve hattı incelemek için sıklıkla kullanılan CANAlyzer programı yardımıyla gözlemlenmiştir. Hatta basılan sinyaller bir database oluşturularak kaydedilmiş ve grafik ekranın gözlemlenmiştir. Bulgular Şekil 3.9'da verilmiştir. Fren pedalına basılma durumu kare dalga biçiminde 1 veya 0 olarak gözlemlenmekle birlikte motor devri ve araç hızının değişken olduğu görülmektedir.



Şekil 3.9. Eğitim seti CANalyzer görüntüsü

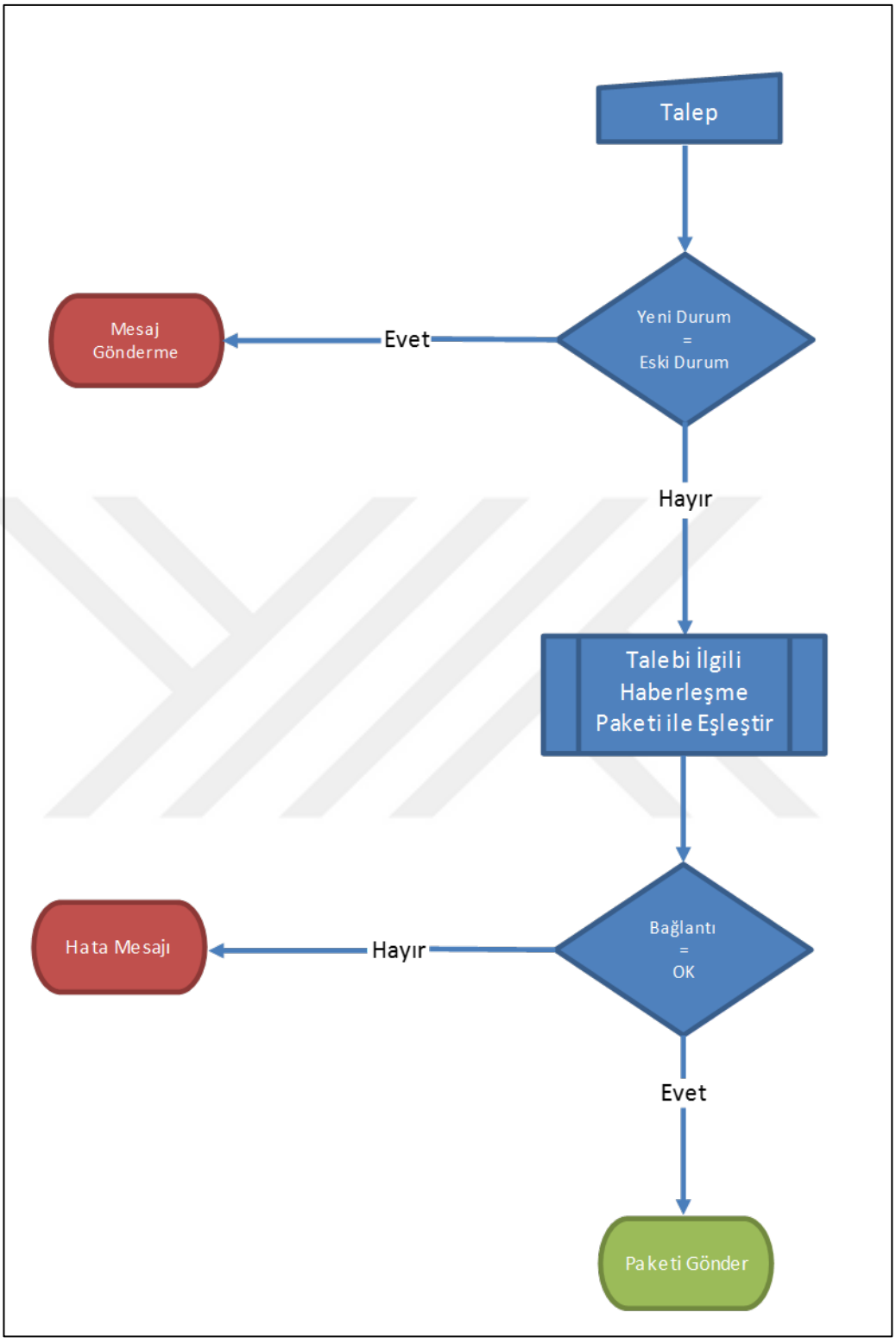
3.6. Geliştirilen Yazılım

Yazılım geliştirme için iki farklı teknolojiden yararlanılmıştır. Bilgisayar tarafındaki arayüz sisteminin geliştirilmesinde .NET platformu kullanılmış ve geliştirme Visual Studio üzerinde yapılmıştır. Gömülü yazılım için ise Arduino'nun kendi geliştirdiği platform kullanılmış ve processing diline benzer bir dil kullanan Arduino arayüzünde yazılım geliştirilmiştir.

CAN haberleşmesi için SparkFun tarafından geliştirilen açık kaynak CAN kütüphanesi [50] kullanılmıştır. Piyasada bulunan diğer açık kaynak kütüphaneler ile kıyaslandığında daha esnek bir yapıya sahip olduğu için bu kütüphane tercih edilmiştir. Aynı zamanda bu kütüphane yine sıklıkla kullanılan ve bulunabilen MCP 2515 entegresini desteklemekte ve onun üzerinde çalışmaktadır.

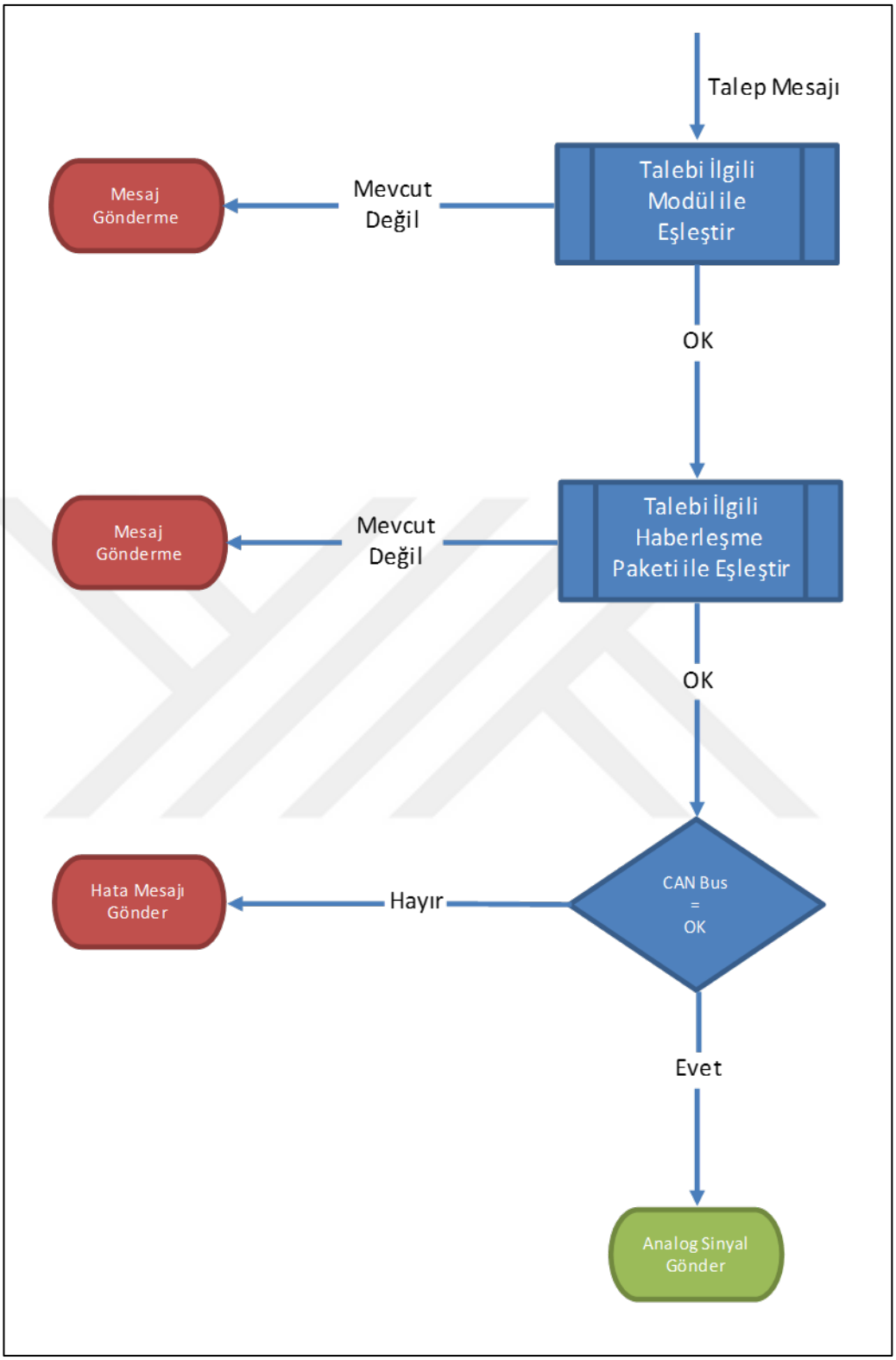
Yazılım içerisinde çalışan algoritma üç adımda düşünülebilir. Bilgisayar üzerinde çalışan arayüz katmanı, transfer modülü katmanı ve haberleşen CAN modülleri katmanı. Bu üçüne dair akış diyagramları aşağıda verilmiştir.

Bilgisayarda çalışan kullanıcı arayüzünün talepleri değerlendirme akış şeması aşağıdaki gibidir (Şekil 3.10).



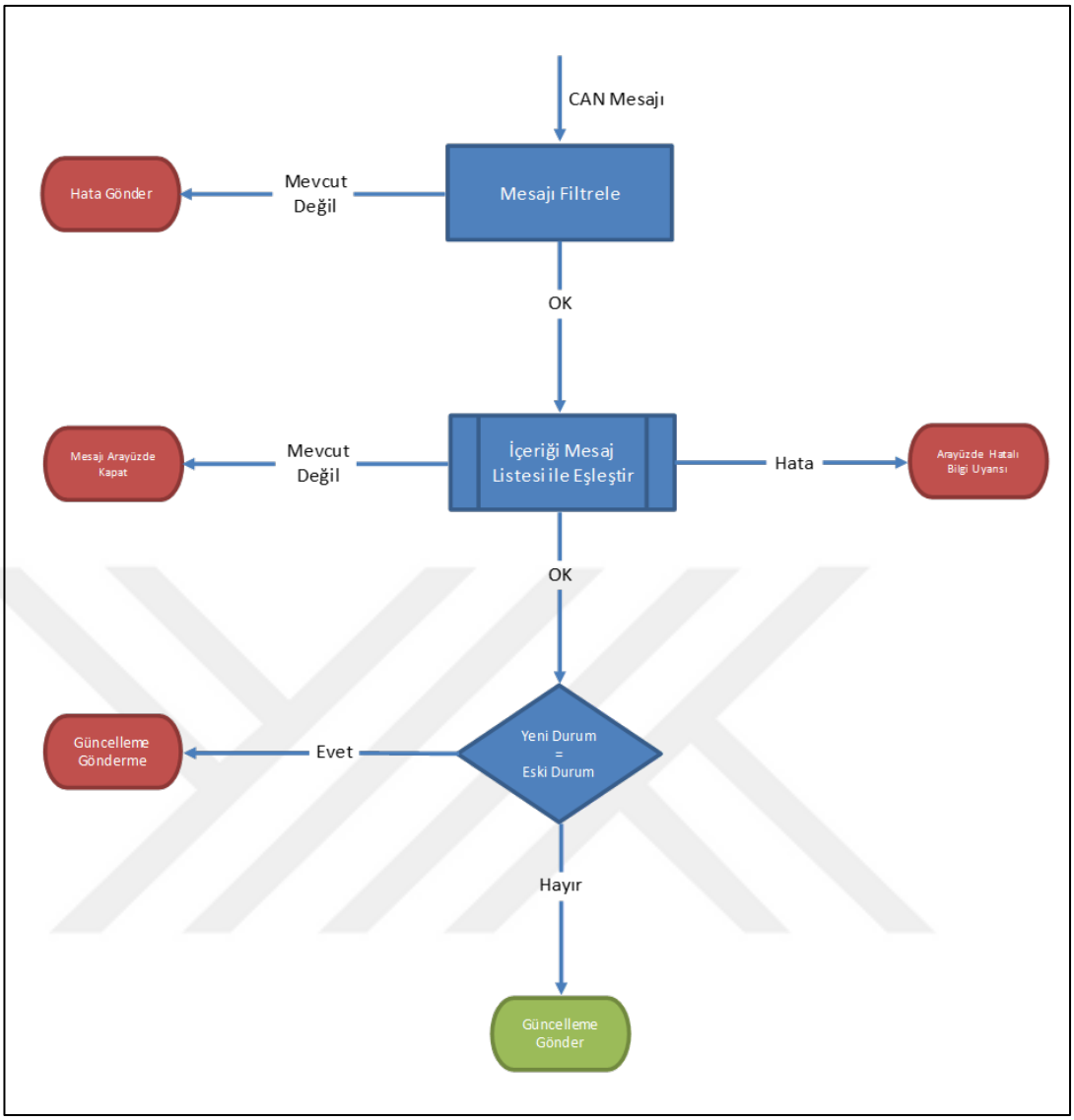
Şekil 3.10. Arayüz talep değerlendirme algoritması

Transfer modülünün bilgisayardan gelen taleplere verdiği tepkiyi gösteren akış şeması aşağıdaki gibidir (Şekil 3.11).



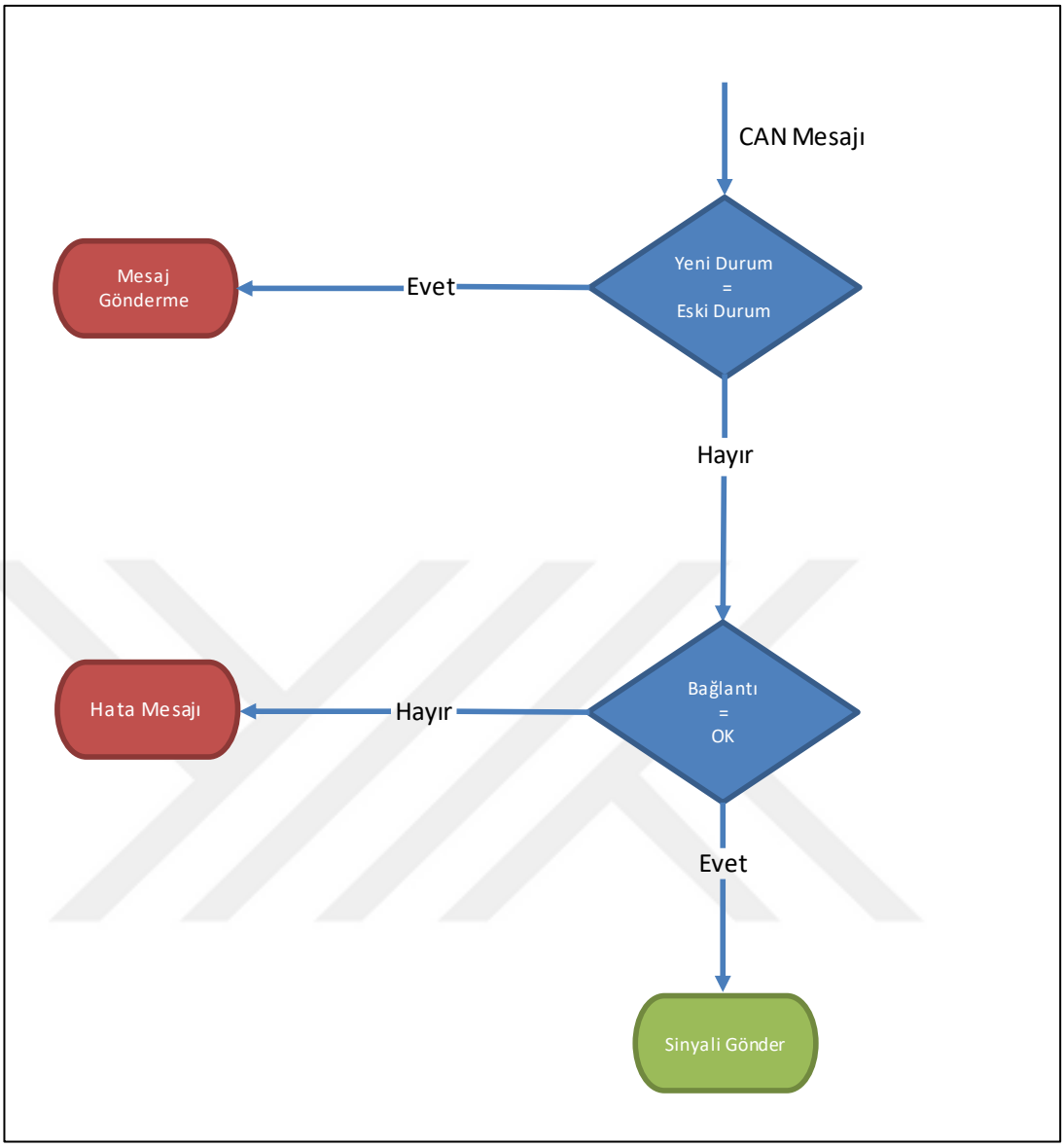
Şekil 3.11. Transfer modülü talep değerlendirme algoritması

Transfer modülünün diğer modüllerden gelen CAN mesajlarına verdiği tepki ve iletim akışını gösteren akış şeması aşağıdaki gibidir (Şekil 3.12).



Şekil 3.12. Transfer modülü CAN mesajı değerlendirme algoritması

CAN modüllerinin transfer modülünden gelen analog sinyallere göre tepkisini gösteren akış şeması aşağıdaki gibidir (Şekil 3.13).



Şekil 3.13. CAN modülleri talep değerlendirme algoritması

4. SONUÇLAR VE ÖNERİLER

Bu tez çalışması kapsamında CAN BUS haberleşmesinin avantajlı yanlarından ve çalışma prensibinden detaylı olarak bahsedilmiş, endüstride kullanılma amacına, literatürdeki örneklerine ve özellikle hatalara dayanıklı yapısına vurgu yapılmıştır. [51] CAN hatlarında haberleşmesinin ne şekilde yapıldığı, mesajların nasıl önceliklendirildiği ve farklı hata durumlarında nasıl tepki verildiğinden bahsedilmiş ve örneklerle açıklanmıştır.

Endüstri 4.0 uygulamalarından bahsedilmiş ve CAN FD yapısı ile birlikte CAN BUS haberleşmesinin Endüstri 4.0 içerisinde kullanımının artması ve önemli bir role sahip olması beklenmektedir. Bununla birlikte Endüstriyel uygulamalarda CAN haberleşmesinin tek eksikliği olan iletimi hızı kavramı karşımıza çıkmaktadır, buna alternatif olarak ise CAN FD ve ilerleyen zamanlarda ethernet haberleşmesi karşımıza çıkmaktadır.

CAN BUS için dört modülden ve bir transfer modülünden oluşan bir eğitim seti hazırlanmış ve eğitim seti içerisinde yapılan çalışmalar belirtilerek, kullanılan yöntemler, fiziksel bağlantılar ve oluşturulan algoritmalar detaylarıyla açıklanmıştır. Hazırlanan CAN eğitim seti ile birlikte CAN haberleşmesinin karakteristik özelliklerinin çalışılabilmesi ve daha fazla araştırmacının CAN BUS hakkında bilgi sahibi olabilmesi hedeflenmektedir.

KAYNAKLAR

[1] Klencke U., Dais S., Lltschel M., Automotive Serial Controller Area Network, *SAE Technical Paper Series,* 1986, **860391**, 1-6.

[2] BOSCH, *CAN Specification,* Version 2.0, Robert Bosch GmbH, Stuttgart, 1991.

[3] Bayılmış, C., IEEE 802.11B Klan Kullanarak Can Segmentleri Genişleten Arabağlaşım Birimi Tasarımı, Doktora Tezi, Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Kocaeli, 2006, 198660.

[4] Karaaslan İ., Afacan T., Demircan E., Başol Ö., Zaim E., İletişim Katmanı Yazılım Mimarisi, *Turkish National Software Engineering Symposium 2013 (UYMS 2013)*, İzmir, Türkiye, 26 Eylül 2013.

[5] Cook J. A., Freudenberg J. S., Controller Area Network (CAN), *EECS 461,* Fall 2008.

[6] Nolte T., Nolin M., Hansson H.A., Real-Time Server-Based Communication with CAN, *IEEE Transactions on Industrial Informatics,* 2005, **1**(3), 192-201.

[7] Tindell K. W., Hansson H., Analysing Real-Time Communications: Controller Area Network (CAN), *Real-Time Systems Symposium, IEEE Xplore,* San Juan, Puerto Rico, 1994.

[8] B.Hallgren, CANbus - hardware description, *ATLAS DCS Workshop,* 2-3 Oct 2000.

[9] Pazul K., Controller Area Network (CAN) Basics, *Microchip Technology Inc.,* 1999, **AN713,** 1-7.

[10] Tindell K., Burns A., Guaranteed Message Latencies For Distributed Safety-Critical Hard Real-Time Control Networks, *1st International CAN Conference*, 1994.

[11] Yang, C., Yao, J., The Design of Distributed Control System Based on CAN Bus, *Electronic and Mechanical Engineering and Information Technology (EMEIT), International Conference,* Heilongjiang, China, 12-14 August 2011.

[12] Corrigan S., Introduction to the Controller Area Network (CAN), *Texas Instruments, Application Report,* **SLOA101B**–August 2002.

[13] Mueller A., Lothspeich T., Plug-and-Secure Communication for CAN, *IEEE International Conference on Communications ICC (ICC 2015)*, London, UK, 8-12 June 2015.

[14] Dinçer E., CAN BUS ile Dağıtık Kontrol Uygulaması, Yüksek Lisans Tezi, Niğde Üniversitesi, Fen Bilimleri Enstitüsü, Niğde, 2010, 271475.

[15] Tindell K., Burns A., Wellings A.J., Calculating Controller Area Network (CAN) Message Response Times, *Control. Eng. Practice,* 1995, **3**(8), 1163-1169.

[16] Natale M.D., Scheduling the CAN BUS with Earliest DeadlineTechniques, *21st IEEE Real-Time Systems Symposium,* 2000, Proceedings., 259-268.

[17] Navet N., Song Y., Simonot-Lion F., Wilwert C., Trends in Automotive Communication Systems, *Proceedings of the IEEE,* 2005, **93**(6), 1204-1223.

[18] Hartwich F., Bassemir A., The Configuration of the CAN Bit Timing, *6th International CAN Conference,* Turin, Italy, 2-4 November 1999.

[19] Lawrenz, W., *CAN System Engineering,* Springer-Verlag, London, UK, 2013.

[20] Dietmayer K., Overberg K., CAN Bit Timing Requirements, *SAE International Congress and Exposition, SAE Technical Paper,* 970295, 1997, https://doi.org/10.4271/970295.

[21] Al-araji S.R., Hussain Z. M., Al-qutayri M.A., *Digital Phase Lock Loops,* Chapter 2, Springer International Publishing, Boston, USA, 2006.

[22] L.B. Fredriksson, CAN for Critical Embedded Automotive Networks, *IEEE Micro 2002*, 28-35.

[23] Nolte T., Hansson H., Norström C., Punnekkat S., Using Bit-Stuffing Distributions in CAN Analysis, IEEE Real-Time Embedded Systems Workshop, 3 December 2001.

[24] Karaca A., Kablolu İletişim Ağlarında Yeni Bir Şifreleme Tabanlı Güvenlik Uygulaması, Doktora Tezi, Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Sakarya, 2012, 330575.

[25] Rufino J., Verissimo P., Arroz G., Almeida C., Rodriguez L., Fault-Tolerant Broadcastsin CAN, *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, IEEE Xplore,* Munich, Germany, 23-25 June 1998.

[26] Anssi S., Tucci-Piergiovanni S., Kuntz S., Gérard S., Terrier F., Enabling Scheduling Analysis for AUTOSAR Systems, *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, Newport Beach, California, USA, 28-31 March 2011.

[27] Short M., Pont M. J., Fault-tolerant, time-triggered communication using CAN, *IEEE Transactions on Industrial Informatics,* 2007, 1-32.

[28] Lam S.S., A Carrier Sense Multiple Access Protocol for Local Networks, *Computer Networks,* 1980, **4**, 21-32.

[29] Emani K.C., Kam K., Zawodniok M., Zheng Y.R., Sarangapani J., Improvement of CAN BUS Performance by using Error-Correction Codes, *Missouri University of Science and Technology, Electrical and Computer Engineering Faculty, Research & Creative Works,* 2007, 1-1-2007.

[30] Lepkowski J., Wolfe B., EMI/ESD Protection Solutions for the CAN Bus, *iCC 2005, CAN in Automation Proceedings,* 2005, 16-24.

[31] Kamuda K., Kalita W., Koodziejski J.F., The Analysis of Signals Propagation in Transmission Lines in the Configuration of CAN-Bus Controller, *iCC 2005, CAN in Automation Proceedings,* 2005, 1-9.

[32] Casarosa G. , Apuzzo M., Fanucci L., Sarti B., Characterization of the EMC Performances of the CAN Bus in a Typical System Bus Architecture for Small Satellites, *Proceedings of the 9th EUROMICRO Conference on Digital System Design,* IEEE, 2006.

[33] Zdenek K., Jiri, S., Simulation of CAN Bus Physical Layer Using SPICE, *Applied Electronics (AE) International Conference,* Pilsen, Czech Republic, 10-12 September 2013.

[34] Dozier R. E., Wildwood, M.O. Beam J.S., Topp E., 2013, U. S. Patent No. US 8,597,054 B2., *U.S. Patent and Trademark Office*.

[35] Wolf M., Weimerskirch A., Paar C., Security in Automotive Bus Systems, *Workshop on Embedded IT-Security in Cars*, Bochum, Germany, November 2004.

[36] Hobson L.B., 2000, U. S. Patent No. 6,122,748, *U.S. Patent and Trademark Office.*

[37] Hartwich F., CAN with Flexible Data-Rate, *iCC 2012, CAN in Automation Proceedings,*2012, 1-9.

[38] McKnight M., IOT, Industry 4.0, Industrial IOT... Why Connected Devices are the Future of Design, *DesTech Conference Proceedings, The International Conference on Design and Technology,* 2017, **2017,** 197-202.

[39] Faul A., Jazdi N., Weyrich M., Approach to Interconnect Existing Industrial Automation Systems with the Industrial Internet, *2016 IEEE,* Berlin, Germany, 6-9 September 2016.

[40] Lasi H., Fettke P., Kemper H.G., Feld T., Hoffmann M., Industry 4.0, *Business & Information Systems Engineer*, 2014, **6**(4), 239–242.

[41] Rüßmann M., Lorenz M., Gerbert P., Waldner M., Justus J., Engel P., Harnisch M., Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries*, Industrial Products & Processes,* April 09, 2015, 1-13.

[42] Plaga S., Tatschner S., Newe T., Logboat – A Simulation Framework Enabling CAN Security Assessments, *2016 IEEE International Conference Proceedings,* 2016, 215-218.

[43] Kim S.H., SeoS.H., KimJ.H., MoonT.Y., Son C.W., Hwang S.H., Jeon J.W., A Gateway System for an Automotive System: LIN, CAN, and FlexRay, *Industrial Informatics, 2008, 6th IEEE International Conference,* Daejeon, South Korea, 13-16 July 2008.

[44] Freiberger S., Albrecht M., Käufl J., Reverse Engineering Technologies for Remanufacturing of Automotive Systems Communicating via CAN Bus, *Journal of Remanufacturing*, 2011, **1**(6), 1-14.

[45] Li R., Liu C., Luo F., A Design for Automotive CAN Bus Monitoring System, *IEEE Vehicle Power and Propulsion Conference (VPPC)*, Harbin, China, 3-5 September 2008.

[46] Geraldo, G., Differences between On Board Diagnostic Systems (EOBD, OBD-II, OBD-BR1 and OBD-BR2), *SAE Technical Paper 2006-01-2671,* 2006, https://doi.org/10.4271/2006-01-2671.

[47] Microchip, Stand-Alone CAN Controller With SPI Interface, MCP2515, Microchip Technology Inc, 2007.

[48] Microchip, High-Speed CAN Transceiver, MCP2551, Microchip Technology Inc, 2010.

[49] https://www.esd-electronics-usa.com (Ziyaret Tarihi: 27.11.2017)

[50] https://creativecommons.orf/licences/by-sa/4.0/ (Ziyaret Tarihi: 25.11.2017)

[51] Hoppe T., Kiltz S., Dittmann J., Security Threatsto Automotive CAN Networks—Practical Examplesand Selected Short-Term Countermeasures, *Reliability Engineering and System Safety*, 2011, **96**, 11–25.

EKLER

Ek-A

```
#include <Canbus.h>
#include <defaults.h>
#include <global.h>
#include <mcp2515.h>
#include <mcp2515_defs.h>
String inputString = "";
boolean stringComplete = false;
String commandString = "";
int flagSenaryo = 0;
// Fren Kontrol Modülü
void setup() {
  Serial.begin(9600);
  delay(1000);
  Canbus.init(CANSPEED_500);  //Initialise MCP2515 CAN controller at the
        specified speed
  delay(1000);
}
void loop()
{
  if(stringComplete)
  {
    stringComplete = false;
    getCommand();
    if(commandString.equals("LED1"))
    {
       flagSenaryo = 0;
    }
    else if(commandString.equals("LED2"))
```

```
  {
   flagSenaryo = 1;
  }
   else if(commandString.equals("LED3"))
  {
   flagSenaryo = 2;
  }
   else if(commandString.equals("LED4"))
  {
   flagSenaryo = 3;
  }
  inputString = "";
}
if(flagSenaryo == 0)
{
   tCAN message;
        message.id = 0x222;
        message.header.rtr = 0;
        message.header.length = 8;
     message.data[0] = 0x00; // Frene basıldı mı
     message.data[1] = 0x00;
     message.data[2] = 0x00;
     message.data[3] = 0x00;
     message.data[4] = 0x00;
     message.data[5] = 0x00;
     message.data[6] = 0x00;
     message.data[7] = 0x00;
     mcp2515_bit_modify(CANCTRL,
      (1<<REQOP2)|(1<<REQOP1)|(1<<REQOP0), 0);
     mcp2515_send_message(&message);
```

```
    delay(500);
    Serial.println("00 00 00 00 00 00 00 00");
}
 else if(flagSenaryo == 1)
{
   tCAN message;
        message.id = 0x222;
        message.header.rtr = 0;
        message.header.length = 8;
    message.data[0] = 0x01; // Frene basıldı mı
    message.data[1] = 0x00;
    message.data[2] = 0x00;
    message.data[3] = 0x00;
    message.data[4] = 0x00;
    message.data[5] = 0x00;
    message.data[6] = 0x00;
    message.data[7] = 0x00;
    mcp2515_bit_modify(CANCTRL,
     (1<<REQOP2)|(1<<REQOP1)|(1<<REQOP0), 0);
    mcp2515_send_message(&message);
    delay(500);
    Serial.println("01 00 00 00 00 00 00 01");
}
 else if(flagSenaryo == 2)
{
   tCAN message;
        message.id = 0x222;
        message.header.rtr = 0;
        message.header.length = 8;
    message.data[0] = 0x02; // Frene basıldı mı
```

```
    message.data[1] = 0x00;
    message.data[2] = 0x00;
    message.data[3] = 0x00;
    message.data[4] = 0x00;
    message.data[5] = 0x00;
    message.data[6] = 0x00;
    message.data[7] = 0x00;
    mcp2515_bit_modify(CANCTRL,
     (1<<REQOP2)|(1<<REQOP1)|(1<<REQOP0), 0);
    mcp2515_send_message(&message);
    delay(500);
    Serial.println("02 00 00 00 00 00 00 02");
}
 else if(flagSenaryo == 3)
{
   tCAN message;
       message.id = 0x222;
       message.header.rtr = 0;
       message.header.length = 8;
    message.data[0] = 0x03; // Frene basıldı mı
    message.data[1] = 0x00;
    message.data[2] = 0x00;
    message.data[3] = 0x00;
    message.data[4] = 0x00;
    message.data[5] = 0x00;
    message.data[6] = 0x00;
    message.data[7] = 0x00;
    mcp2515_bit_modify(CANCTRL,
     (1<<REQOP2)|(1<<REQOP1)|(1<<REQOP0), 0);
    mcp2515_send_message(&message);
    delay(500);
```

```
        Serial.println("03 00 00 00 00 00 00 03");
  }
}
void getCommand()
{
  if(inputString.length()>0)
  {
     commandString = inputString.substring(1,5);
  }
}
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Ek-B

```
#include <Canbus.h>
#include <defaults.h>
#include <global.h>
#include <mcp2515.h>
#include <mcp2515_defs.h>
#include <LiquidCrystal.h>
String inputString = "";
boolean stringComplete = false;
String commandString = "";
int flagSenaryo = 0;
byte aracHizi = 100;
byte motorDevri = 140;
const int potPin = A2; //pin A0 to read analog input
int value; //save analog value
LiquidCrystal lcd(8,9,4,5,6,7);
// Motor Kontrol Modülü
void setup() {
  Serial.begin(9600);
  Serial.println("CAN Write - Testing transmission of CAN Bus messages");
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("Motor Kontrol");
  if (Canbus.init(CANSPEED_500)) //Initialise MCP2515 CAN controller at the
        specified speed
    Serial.println("CAN Init ok");
  else
    Serial.println("Can't init CAN");
}
```

```
void loop()
{
 tCAN message_read;
 if (mcp2515_check_message())
 {
  if (mcp2515_get_message(&message_read))
  {
   if (message_read.data[0] == 0x01)
   {
    flagSenaryo = 1;
   }
   else if(message_read.data[0] == 0x02)
   {
    flagSenaryo = 2;
   }
   else if(message_read.data[0] == 0x03)
   {
    flagSenaryo = 3;
   }
   else
   {
    flagSenaryo = 0;
   }
  }
 }
 if (flagSenaryo == 1)
 {
  if (aracHizi > 0)
   aracHizi=aracHizi-1;
  if (motorDevri > 24)
```

```
  motorDevri = motorDevri - 4;
  tCAN message;
  message.id = 0x111; //formatted in HEX
  message.header.rtr = 0;
  message.header.length = 8; //formatted in DEC
  message.data[0] = motorDevri; // Motor Devri - Çözünürlük 25 - Min:0 -
      Max:5000
  message.data[1] = aracHizi; // Araç Hızı - Çözünürlük 1 - Min:0 - Max:250
  message.data[2] = 0x3F; // Gaz Pedalı Yüzdesi - Çözünürlük 1 - Min:0 - Max:100
  message.data[3] = 0x04; // Şanzıman bilgisi
  message.data[4] = 0x00;
  message.data[5] = 0x00;
  message.data[6] = 0x00;
  message.data[7] = 0x00;
  mcp2515_bit_modify(CANCTRL, (1 << REQOP2) | (1 << REQOP1) | (1 <<
      REQOP0), 0);
  mcp2515_send_message(&message);
  lcd.setCursor(0,0);
  lcd.print("Frene Basildi");
  lcd.setCursor(0,1);
  for(int i=0;i<message.header.length;i++)
  {
     lcd.print(message.data[i],HEX);
  }
  delay(500);
}
else if(flagSenaryo == 2)
{
  aracHizi = 100;
  motorDevri = 180;
  tCAN message;
```

```
  message.id = 0x111; //formatted in HEX
  message.header.rtr = 0;
  message.header.length = 8; //formatted in DEC
  message.data[0] = motorDevri; // Motor Devri - Çözünürlük 25 - Min:0 -
      Max:5000
  message.data[1] = aracHizi; // Araç Hızı - Çözünürlük 1 - Min:0 - Max:250
  message.data[2] = 0x3F; // Gaz Pedalı Yüzdesi - Çözünürlük 1 - Min:0 - Max:100
  message.data[3] = 0x03; // Şanzıman bilgisi
  message.data[4] = 0x00;
  message.data[5] = 0x00;
  message.data[6] = 0x00;
  message.data[7] = 0x00;
  mcp2515_bit_modify(CANCTRL, (1 << REQOP2) | (1 << REQOP1) | (1 <<
      REQOP0), 0);
  mcp2515_send_message(&message);
  lcd.setCursor(0,0);
  lcd.print("Vites Azaltma");
  lcd.setCursor(0,1);
  for(int i=0;i<message.header.length;i++)
  {
     lcd.print(message.data[i],HEX);
  }
  delay(500);
}
else if(flagSenaryo == 3)
{
  aracHizi = 100;
  motorDevri = 100;
  tCAN message;
  message.id = 0x111; //formatted in HEX
  message.header.rtr = 0;
```

```
  message.header.length = 8; //formatted in DEC
  message.data[0] = motorDevri; // Motor Devri - Çözünürlük 25 - Min:0 -
      Max:5000
  message.data[1] = aracHizi; // Araç Hızı - Çözünürlük 1 - Min:0 - Max:250
  message.data[2] = 0x3F; // Gaz Pedalı Yüzdesi - Çözünürlük 1 - Min:0 - Max:100
  message.data[3] = 0x05; // Şanzıman bilgisi
  message.data[4] = 0x00;
  message.data[5] = 0x00;
  message.data[6] = 0x00;
  message.data[7] = 0x00;
  mcp2515_bit_modify(CANCTRL, (1 << REQOP2) | (1 << REQOP1) | (1 <<
      REQOP0), 0);
  mcp2515_send_message(&message);
  lcd.setCursor(0,0);
  lcd.print("Vites Arttirma");
  lcd.setCursor(0,1);
  for(int i=0;i<message.header.length;i++)
  {
     lcd.print(message.data[i],HEX);
  }
  delay(500);
}
else
{
  aracHizi = 100;
  value = analogRead(potPin);
  value = map(value, 0, 1023, 0, 255);
  tCAN message;
  message.id = 0x111; //formatted in HEX
  message.header.rtr = 0;
  message.header.length = 8; //formatted in DEC
```

```
    message.data[0] = 100; // Motor Devri - Çözünürlük 25 - Min:0 - Max:5000
    message.data[1] = 0x0F; // Araç Hızı - Çözünürlük 1 - Min:0 - Max:250
    message.data[2] = value; // Gaz Pedalı Yüzdesi - Çözünürlük 1 - Min:0 - Max:100
    message.data[3] = 0x04;
    message.data[4] = 0x00;
    message.data[5] = 0x00;
    message.data[6] = 0x00;
    message.data[7] = 0x00;
    mcp2515_bit_modify(CANCTRL, (1 << REQOP2) | (1 << REQOP1) | (1 <<
        REQOP0), 0);
    mcp2515_send_message(&message);
    delay(500);
    lcd.setCursor(0,0);
    lcd.print("Motor Kontrol");
    lcd.setCursor(0,1);
    for(int i=0;i<message.header.length;i++)
    {
       lcd.print(message.data[i],HEX);
    }
  }
}
```

Ek-C

```
#include <Canbus.h>
#include <defaults.h>
#include <global.h>
#include <mcp2515.h>
#include <mcp2515_defs.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(8,9,4,5,6,7);
void setup() {
  Serial.begin(9600); // For debug use
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("Motor Kontrol");
  delay(1000);
  Canbus.init(CANSPEED_500);
  delay(1000);
}
void loop(){
tCAN message;
if (mcp2515_check_message())
        {
   if (mcp2515_get_message(&message))
        {
       if(message.id == 0x111)
           {
             lcd.setCursor(0,1);
             for(int i=0;i<message.header.length;i++)
             {
               lcd.print(message.data[i],HEX);
```

```
      lcd.print(" ");
    }
    delay(500);
  }
  if(message.id == 0x222)
  {
    lcd.setCursor(0,0);
    if (message.data[0] == 0x01)
    {
      lcd.print("Frene Basildi");
    }
    else if(message.data[0] == 0x02)
    {
      lcd.print("Vites Azaltma");
    }
    else if(message.data[0] == 0x03)
    {
      lcd.print("Vites Arttirma");
    }
    else
    {
      lcd.print("Motor Kontrol");
    }
    delay(500);
  }
 }}
}
```

Ek-D

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CANBUSControllerInterface
{
    public partial class Form3 : Form
    {
        // Hata Çerçevesi Simülasyonu Değişkenler
        Timer hataCercevesiTimer = new Timer();
        Boolean hataCercevesiTimerCount = true;
        Color labelModul1Color = Color.ForestGreen;
        Color labelModul2Color = Color.ForestGreen;
        Color labelModul3Color = Color.ForestGreen;
        Color labelModul4Color = Color.ForestGreen;
        Color labelModul5Color = Color.ForestGreen;
        Color labelModul6Color = Color.ForestGreen;
        Color labelModul1SecondColor = Color.Black;
        Color labelModul2SecondColor = Color.Black;
        Color labelModul3SecondColor = Color.Black;
        Color labelModul4SecondColor = Color.Black;
```

```
Color labelModul5SecondColor = Color.Black;
Color labelModul6SecondColor = Color.Black;
// TEC REC Simülasyonu Değişkenler
Random rnd = new Random();
Timer tecRecTimer = new Timer();
int TEC1;
int TEC2;
int TEC3;
int REC1;
int REC2;
int REC3;
int modul1Durum;
int modul2Durum;
int modul3Durum;
Color modul1NameColor = Color.ForestGreen;
Color modul2NameColor = Color.ForestGreen;
Color modul3NameColor = Color.ForestGreen;
Boolean tecRecTimerCount = true;
// Mesaj Önceliklendirme Simülasyonu Değişkenler
Timer onceliklendirmeTimer1 = new Timer();
int onceliklendirmeTimerCounter1;
Timer onceliklendirmeTimer2 = new Timer();
int onceliklendirmeTimerCounter2;
// Hat Yoğunluğu Hesaplama Değişkenler
List<int> repTimes;
// Real Time Simülasyon Değişkenler
bool isConnected = false;
String[] ports;
SerialPort port;
Timer senaryo2Timer = new Timer();
```

```
int hiz;
int devir;
public Form3()
{
   InitializeComponent();
   //Hata Çerçevesi Simülasyonu
   hataCercevesiTimer.Interval = (750);
   hataCercevesiTimer.Tick += new EventHandler(hataCercevesiTimer_Tick);
   hataCercevesiTimer.Start();

   // TEC REC Simülasyonu
   tecRecTimer.Interval = (750);
   tecRecTimer.Tick += new EventHandler(tecRecTimer_Tick);
   tecRecTimer.Start();
   TEC1 = rnd.Next(119, 127);
   TEC2 = rnd.Next(119, 127);
   TEC3 = rnd.Next(119, 127);
   REC1 = rnd.Next(119, 127);
   REC2 = rnd.Next(119, 127);
   REC3 = rnd.Next(119, 127);
   modul1Durum = 0;
   modul2Durum = 0;
   modul3Durum = 0;
   labelTecRecDoldur();
   // Mesaj Önceliklendirme Simülasyonu
   onceliklendirmeTimer1.Interval = (1500);
   onceliklendirmeTimer1.Tick += new
EventHandler(onceliklendirmeTimer1_Tick);
   onceliklendirmeTimerCounter1 = 0;
   onceliklendirmeTimer2.Interval = (1500);
```

```
    onceliklendirmeTimer2.Tick += new
 EventHandler(onceliklendirmeTimer2_Tick);
    onceliklendirmeTimerCounter2 = 0;
    // Hat Yoğunluğu Hesaplama
    repTimes = new List<int>();
    cmbCycleTime.SelectedIndex = 0;
    cmbHatHizi.SelectedIndex = 0;
    // Real Time Simülasyon
    getAvailableComPorts();
    foreach (string port in ports)
    {
        cmbPortSecimi.Items.Add(port);
        if (ports[0] != null)
        {
            cmbPortSecimi.SelectedItem = ports[0];
        }
    }
    senaryo2Timer.Interval = (300);
    senaryo2Timer.Tick += new EventHandler(senaryo2Timer_Tick);
}
private void tecRecTimer_Tick(object sender, EventArgs e)
{
    if (tecRecTimerCount)
    {
        lblModul1Name.ForeColor = modul1NameColor;
        lblModul2Name.ForeColor = modul2NameColor;
        lblModul3Name.ForeColor = modul3NameColor;
    }
    else
    {
```

```
            lblModul1Name.ForeColor = Color.Black;
            lblModul2Name.ForeColor = Color.Black;
            lblModul3Name.ForeColor = Color.Black;
        }
        tecRecTimerCount = !tecRecTimerCount;
    }
    private void hataCercevesiTimer_Tick(object sender, EventArgs e)
    {
        if (hataCercevesiTimerCount)
        {
            lblModul1.ForeColor = labelModul1SecondColor;
            lblModul2.ForeColor = labelModul2SecondColor;
            lblModul3.ForeColor = labelModul3SecondColor;
            lblModul4.ForeColor = labelModul4SecondColor;
            lblModul5.ForeColor = labelModul5SecondColor;
            lblModul6.ForeColor = labelModul6SecondColor;
        }
        else
        {
            lblModul1.ForeColor = labelModul1Color;
            lblModul2.ForeColor = labelModul2Color;
            lblModul3.ForeColor = labelModul3Color;
            lblModul4.ForeColor = labelModul4Color;
            lblModul5.ForeColor = labelModul5Color;
            lblModul6.ForeColor = labelModul6Color;
        }
        hataCercevesiTimerCount = !hataCercevesiTimerCount;
    }
    private void onceliklendirmeTimer1_Tick(object sender, EventArgs e)
    {
```

```
switch (onceliklendirmeTimerCounter1)
{
   case 0:
      lblOncelik11.Visible = true;
      lblOncelik12.Visible = true;
      lblOncelik13.Visible = true;
      lblOncelik14.Visible = true;
      lblOncelik11.ForeColor = Color.Green;
      lblOncelik12.ForeColor = Color.Green;
      lblOncelik13.ForeColor = Color.Green;
      lblOncelik14.ForeColor = Color.Green;
      lblOncelik11.Text = "0";
      lblOncelik12.Text = "0";
      lblOncelik13.Text = "0";
      lblOncelik14.Text = "0";
      label8.ForeColor = Color.Black;
      label11.ForeColor = Color.Black;
      label12.ForeColor = Color.Black;
      label20.ForeColor = Color.Black;
      break;
   case 1:
      lblOncelik21.Visible = true;
      lblOncelik22.Visible = true;
      lblOncelik23.Visible = true;
      lblOncelik24.Visible = true;
      lblOncelik21.ForeColor = Color.Green;
      lblOncelik22.ForeColor = Color.Green;
      lblOncelik23.ForeColor = Color.Green;
      lblOncelik24.ForeColor = Color.Green;
      lblOncelik21.Text = "1";
```

```
            lblOncelik22.Text = "1";
            lblOncelik23.Text = "1";
            lblOncelik24.Text = "1";
            break;
        case 2:
            lblOncelik31.Visible = true;
            lblOncelik32.Visible = true;
            lblOncelik33.Visible = true;
            lblOncelik34.Visible = true;
            lblOncelik31.ForeColor = Color.Green;
            lblOncelik32.ForeColor = Color.Green;
            lblOncelik33.ForeColor = Color.Green;
            lblOncelik34.ForeColor = Color.Green;
            lblOncelik31.Text = "0";
            lblOncelik32.Text = "0";
            lblOncelik33.Text = "0";
            lblOncelik34.Text = "0";
            break;
        case 3:
            lblOncelik41.Visible = true;
            lblOncelik42.Visible = true;
            lblOncelik43.Visible = true;
            lblOncelik44.Visible = true;
            lblOncelik41.ForeColor = Color.Green;
            lblOncelik42.ForeColor = Color.Red;
            lblOncelik43.ForeColor = Color.Green;
            lblOncelik44.ForeColor = Color.Green;
            lblOncelik41.Text = "0";
            lblOncelik42.Text = "1";
            lblOncelik43.Text = "0";
```

```
            lblOncelik44.Text = "0";
            break;
        case 4:
            lblOncelik51.Visible = true;
            lblOncelik52.Visible = true;
            lblOncelik53.Visible = true;
            lblOncelik54.Visible = true;
            lblOncelik51.ForeColor = Color.Green;
            lblOncelik52.ForeColor = Color.Red;
            lblOncelik53.ForeColor = Color.Green;
            lblOncelik54.ForeColor = Color.Green;
            lblOncelik51.Text = "1";
            lblOncelik52.Text = "1";
            lblOncelik53.Text = "1";
            lblOncelik54.Text = "1";
            break;
        case 5:
            lblOncelik61.Visible = true;
            lblOncelik62.Visible = true;
            lblOncelik63.Visible = true;
            lblOncelik64.Visible = true;
            lblOncelik61.ForeColor = Color.Red;
            lblOncelik62.ForeColor = Color.Red;
            lblOncelik63.ForeColor = Color.Green;
            lblOncelik64.ForeColor = Color.Green;
            lblOncelik61.Text = "1";
            lblOncelik62.Text = "0";
            lblOncelik63.Text = "0";
            lblOncelik64.Text = "0";
            break;
```

```
case 6:
    lblOncelik71.Visible = true;
    lblOncelik72.Visible = true;
    lblOncelik73.Visible = true;
    lblOncelik74.Visible = true;
    lblOncelik71.ForeColor = Color.Red;
    lblOncelik72.ForeColor = Color.Red;
    lblOncelik73.ForeColor = Color.Green;
    lblOncelik74.ForeColor = Color.Green;
    lblOncelik71.Text = "1";
    lblOncelik72.Text = "0";
    lblOncelik73.Text = "0";
    lblOncelik74.Text = "0";
    break;
case 7:
    lblOncelik81.Visible = true;
    lblOncelik82.Visible = true;
    lblOncelik83.Visible = true;
    lblOncelik84.Visible = true;
    lblOncelik81.ForeColor = Color.Red;
    lblOncelik82.ForeColor = Color.Red;
    lblOncelik83.ForeColor = Color.Green;
    lblOncelik84.ForeColor = Color.Red;
    lblOncelik81.Text = "1";
    lblOncelik82.Text = "0";
    lblOncelik83.Text = "0";
    lblOncelik84.Text = "1";
    break;
case 8:
    label8.ForeColor = Color.Red;
```

```
            label11.ForeColor = Color.Red;
            label12.ForeColor = Color.Green;
            label20.ForeColor = Color.Red;
            onceliklendirmeTimer1.Stop();
            break;
        default:
            break;
    }
    onceliklendirmeTimerCounter1++;
}
private void onceliklendirmeTimer2_Tick(object sender, EventArgs e)
{
    switch (onceliklendirmeTimerCounter2)
    {
        case 0:
            lblOncelik11.Visible = true;
            lblOncelik12.Visible = true;
            lblOncelik13.Visible = true;
            lblOncelik14.Visible = true;
            lblOncelik11.ForeColor = Color.Green;
            lblOncelik12.ForeColor = Color.Green;
            lblOncelik13.ForeColor = Color.Green;
            lblOncelik14.ForeColor = Color.Green;
            lblOncelik11.Text = "0";
            lblOncelik12.Text = "0";
            lblOncelik13.Text = "0";
            lblOncelik14.Text = "0";
            label8.ForeColor = Color.Black;
            label11.ForeColor = Color.Black;
            label12.ForeColor = Color.Black;
```

```
    label20.ForeColor = Color.Black;
    break;
case 1:
    lblOncelik21.Visible = true;
    lblOncelik22.Visible = true;
    lblOncelik23.Visible = true;
    lblOncelik24.Visible = true;
    lblOncelik21.ForeColor = Color.Green;
    lblOncelik22.ForeColor = Color.Green;
    lblOncelik23.ForeColor = Color.Green;
    lblOncelik24.ForeColor = Color.Green;
    lblOncelik21.Text = "0";
    lblOncelik22.Text = "0";
    lblOncelik23.Text = "0";
    lblOncelik24.Text = "0";
    break;
case 2:
    lblOncelik31.Visible = true;
    lblOncelik32.Visible = true;
    lblOncelik33.Visible = true;
    lblOncelik34.Visible = true;
    lblOncelik31.ForeColor = Color.Green;
    lblOncelik32.ForeColor = Color.Green;
    lblOncelik33.ForeColor = Color.Green;
    lblOncelik34.ForeColor = Color.Red;
    lblOncelik31.Text = "0";
    lblOncelik32.Text = "0";
    lblOncelik33.Text = "0";
    lblOncelik34.Text = "1";
    break;
```

```
case 3:
    lblOncelik41.Visible = true;
    lblOncelik42.Visible = true;
    lblOncelik43.Visible = true;
    lblOncelik44.Visible = true;
    lblOncelik41.ForeColor = Color.Green;
    lblOncelik42.ForeColor = Color.Green;
    lblOncelik43.ForeColor = Color.Green;
    lblOncelik44.ForeColor = Color.Red;
    lblOncelik41.Text = "1";
    lblOncelik42.Text = "1";
    lblOncelik43.Text = "1";
    lblOncelik44.Text = "1";
    break;
case 4:
    lblOncelik51.Visible = true;
    lblOncelik52.Visible = true;
    lblOncelik53.Visible = true;
    lblOncelik54.Visible = true;
    lblOncelik51.ForeColor = Color.Green;
    lblOncelik52.ForeColor = Color.Green;
    lblOncelik53.ForeColor = Color.Green;
    lblOncelik54.ForeColor = Color.Red;
    lblOncelik51.Text = "0";
    lblOncelik52.Text = "0";
    lblOncelik53.Text = "0";
    lblOncelik54.Text = "0";
    break;
case 5:
    lblOncelik61.Visible = true;
```

```
            lblOncelik62.Visible = true;
            lblOncelik63.Visible = true;
            lblOncelik64.Visible = true;
            lblOncelik61.ForeColor = Color.Red;
            lblOncelik62.ForeColor = Color.Green;
            lblOncelik63.ForeColor = Color.Red;
            lblOncelik64.ForeColor = Color.Red;
            lblOncelik61.Text = "1";
            lblOncelik62.Text = "0";
            lblOncelik63.Text = "1";
            lblOncelik64.Text = "0";
            break;
        case 6:
            lblOncelik71.Visible = true;
            lblOncelik72.Visible = true;
            lblOncelik73.Visible = true;
            lblOncelik74.Visible = true;
            lblOncelik71.ForeColor = Color.Red;
            lblOncelik72.ForeColor = Color.Green;
            lblOncelik73.ForeColor = Color.Red;
            lblOncelik74.ForeColor = Color.Red;
            lblOncelik71.Text = "1";
            lblOncelik72.Text = "1";
            lblOncelik73.Text = "0";
            lblOncelik74.Text = "0";
            break;
        case 7:
            lblOncelik81.Visible = true;
            lblOncelik82.Visible = true;
            lblOncelik83.Visible = true;
```

```
            lblOncelik84.Visible = true;
            lblOncelik81.ForeColor = Color.Red;
            lblOncelik82.ForeColor = Color.Green;
            lblOncelik83.ForeColor = Color.Red;
            lblOncelik84.ForeColor = Color.Red;
            lblOncelik81.Text = "1";
            lblOncelik82.Text = "0";
            lblOncelik83.Text = "0";
            lblOncelik84.Text = "1";
            break;
        case 8:
            onceliklendirmeTimer2.Stop();
            label8.ForeColor = Color.Red;
            label11.ForeColor = Color.Green;
            label12.ForeColor = Color.Red;
            label20.ForeColor = Color.Red;
            break;
        default:
            break;
    }
    onceliklendirmeTimerCounter2++;
}

// Hata Çerçevesi Simülasyonu Butonlar
private void btnHataYaptir_Click(object sender, EventArgs e)
{
    int a = cmbHataCercevesiModuller.SelectedIndex;
    switch (a)
    {
        case 0:
```

```
            labelModul1Color = Color.Red;
            labelModul1SecondColor = Color.Orange;
            lblModul1.Text = "Hata Çerçevesi";
            break;
        case 1:
            labelModul2Color = Color.Red;
            labelModul2SecondColor = Color.Orange;
            lblModul2.Text = "Hata Çerçevesi";
            break;
        case 2:
            labelModul3Color = Color.Red;
            labelModul3SecondColor = Color.Orange;
            lblModul3.Text = "Hata Çerçevesi";
            break;
        case 3:
            labelModul4Color = Color.Red;
            labelModul4SecondColor = Color.Orange;
            lblModul4.Text = "Hata Çerçevesi";
            break;
        case 4:
            labelModul5Color = Color.Red;
            labelModul5SecondColor = Color.Orange;
            lblModul5.Text = "Hata Çerçevesi";
            break;
        case 5:
            labelModul6Color = Color.Red;
            labelModul6SecondColor = Color.Orange;
            lblModul6.Text = "Hata Çerçevesi";
            break;
        default:
```

```
            break;
        }
    }
    private void btnHataSil_Click(object sender, EventArgs e)
    {
        int a = cmbHataCercevesiModuller.SelectedIndex;
        switch (a)
        {
            case 0:
                labelModul1Color = Color.ForestGreen;
                labelModul1SecondColor = Color.Black;
                lblModul1.Text = "Normal Çerçeve";
                break;
            case 1:
                labelModul2Color = Color.ForestGreen;
                labelModul3SecondColor = Color.Black;
                lblModul2.Text = "Normal Çerçeve";
                break;
            case 2:
                labelModul3Color = Color.ForestGreen;
                labelModul3SecondColor = Color.Black;
                lblModul3.Text = "Normal Çerçeve";
                break;
            case 3:
                labelModul4Color = Color.ForestGreen;
                labelModul4SecondColor = Color.Black;
                lblModul4.Text = "Normal Çerçeve";
                break;
            case 4:
                labelModul5Color = Color.ForestGreen;
```

```
            labelModul5SecondColor = Color.Black;
            lblModul5.Text = "Normal Çerçeve";
            break;
        case 5:
            labelModul6Color = Color.ForestGreen;
            labelModul6SecondColor = Color.Black;
            lblModul6.Text = "Normal Çerçeve";
            break;
        default:
            break;
    }
}
private void btnHataTemizle_Click(object sender, EventArgs e)
{
    labelModul1Color = Color.ForestGreen;
    labelModul1SecondColor = Color.Black;
    lblModul1.Text = "Normal Çerçeve";
    labelModul2Color = Color.ForestGreen;
    labelModul3SecondColor = Color.Black;
    lblModul2.Text = "Normal Çerçeve";
    labelModul3Color = Color.ForestGreen;
    labelModul3SecondColor = Color.Black;
    lblModul3.Text = "Normal Çerçeve";
    labelModul4Color = Color.ForestGreen;
    labelModul4SecondColor = Color.Black;
    lblModul4.Text = "Normal Çerçeve";
    labelModul5Color = Color.ForestGreen;
    labelModul5SecondColor = Color.Black;
    lblModul5.Text = "Normal Çerçeve";
    labelModul6Color = Color.ForestGreen;
```

```
    labelModul6SecondColor = Color.Black;
    lblModul6.Text = "Normal Çerçeve";
}
private void btnHataCercevesiBilgi_Click(object sender, EventArgs e)
{
    lblHataCercevesiBilgi.Visible = !lblHataCercevesiBilgi.Visible;
}
// TEC REC Simülasyonu Butonlar ve Fonksiyonlar
private void btnHattiSifirla_Click(object sender, EventArgs e)
{
    TEC1 = rnd.Next(119, 127);
    TEC2 = rnd.Next(119, 127);
    TEC3 = rnd.Next(119, 127);
    REC1 = rnd.Next(119, 127);
    REC2 = rnd.Next(119, 127);
    REC3 = rnd.Next(119, 127);
    modul1Durum = 0;
    modul2Durum = 0;
    modul3Durum = 0;
    labelTecRecDoldur();
}
private void labelTecRecDoldur()
{
    lblModul1TEC.Text = TEC1.ToString();
    lblModul1REC.Text = REC1.ToString();
    lblModul2TEC.Text = TEC2.ToString();
    lblModul2REC.Text = REC2.ToString();
    lblModul3TEC.Text = TEC3.ToString();
    lblModul3REC.Text = REC3.ToString();
}
```

```
private void btnHatasizMesajGonder_Click(object sender, EventArgs e)
{
   int seciliModul = cmbNormalCalismaModulSecimi.SelectedIndex;
   switch (seciliModul)
   {
      case 0:
         if (modul1Durum != 2)
         {
            TEC1--;
            if (REC2 > 127)
            {
               REC2 = rnd.Next(119, 127);
            }
            else if (REC2 > 0)
            {
               REC2--;
            }
            if (REC3 > 127)
            {
               REC3 = rnd.Next(119, 127);
            }
            else if (REC3 > 0)
            {
               REC3--;
            }
         }
         else
         {
            MessageBox.Show("Modül Bus Off durumundayken mesaj basamaz",
"Modül Bus Off Hatası");
```

```
            }
            break;
        case 1:
            if (modul2Durum != 2)
            {
                TEC2--;
                if (REC1 > 127)
                {
                    REC1 = rnd.Next(119, 127);
                }
                else if (REC1 > 0)
                {
                    REC1--;
                }
                if (REC3 > 127)
                {
                    REC3 = rnd.Next(119, 127);
                }
                else if (REC3 > 0)
                {
                    REC3--;
                }
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj basamaz",
"Modül Bus Off Hatası");
            }
            break;
        case 2:
```

```
            if (modul3Durum != 2)
            {
                TEC3--;
                if (REC2 > 127)
                {
                    REC2 = rnd.Next(119, 127);
                }
                else if (REC2 > 0)
                {
                    REC2--;
                }
                if (REC1 > 127)
                {
                    REC1 = rnd.Next(119, 127);
                }
                else if (REC1 > 0)
                {
                    REC1--;
                }
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj basamaz",
"Modül Bus Off Hatası");
            }
            break;
        default:
            MessageBox.Show("Lütfen Modül Seçin", "Modül Seçilmedi");
            break;
    }
```

```
    labelTecRecDoldur();
}
private void btnHataGonder_Click(object sender, EventArgs e)
{
    int seciliHata = cmbHataGonderHataSecimi.SelectedIndex;
    int seciliModul = cmbHataGonderModulSecimi.SelectedIndex;
    switch (seciliHata)
    {
        case 0: // Bit hatası
            switch (seciliModul)
            {
                case 0:
                    if (modul1Durum != 2)
                    {
                        REC2 += 8;
                        REC3 += 8;
                        TEC1 += 8;
                    }
                    else
                    {
                        MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
                    }
                    break;
                case 1:
                    if (modul2Durum != 2)
                    {
                        REC1 += 8;
                        REC3 += 8;
                        TEC2 += 8;
```

```
                }
                else
                {
                    MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
                }
                break;
            case 2:
                if (modul3Durum != 2)
                {
                    REC2 += 8;
                    REC1 += 8;
                    TEC3 += 8;
                }
                else
                {
                    MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
                }
                break;
            default:
                MessageBox.Show("Lütfen Modül Seçin", "Modül Seçilmedi");
                break;
        }
        break;
    case 1: // ACK Hatası
        switch (seciliModul)
        {
            case 0:
                if (modul1Durum != 2)
                {
```

```
                REC2 += 1;
                REC3 += 1;
                if (modul1Durum != 1)
                    TEC1 += 8;
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
            }
            break;
        case 1:
            if (modul2Durum != 2)
            {
                REC1 += 1;
                REC3 += 1;
                if (modul2Durum != 1)
                    TEC2 += 8;
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
            }
            break;
        case 2:
            if (modul3Durum != 2)
            {
                REC2 += 1;
                REC1 += 1;
                if (modul3Durum != 1)
```

```
                TEC3 += 8;
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj basamaz", "Modül Bus Off Hatası");
            }
            break;
        default:
            MessageBox.Show("Lütfen Modül Seçin", "Modül Seçilmedi");
            break;
    }
    break;
case 2: // Bit Stuffing Hatası
    switch (seciliModul)
    {
        case 0:
            if (modul1Durum != 2)
            {
                REC2 += 1;
                REC3 += 1;
                TEC1 += 8;
            }
            else
            {
                MessageBox.Show("Modül Bus Off durumundayken mesaj basamaz", "Modül Bus Off Hatası");
            }
            break;
        case 1:
            if (modul2Durum != 2)
```

```
                {
                    REC1 += 1;
                    REC3 += 1;
                    TEC2 += 8;
                }
                else
                {
                    MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
                }
                break;
            case 2:
                if (modul3Durum != 2)
                {
                    REC2 += 1;
                    REC1 += 1;
                    TEC3 += 8;
                }
                else
                {
                    MessageBox.Show("Modül Bus Off durumundayken mesaj
basamaz", "Modül Bus Off Hatası");
                }
                break;
            default:
                MessageBox.Show("Lütfen Modül Seçin", "Modül Seçilmedi");
                break;
        }
        break;
    default:
        MessageBox.Show("Lütfen Hata Tipi Seçin","Hata Tipi Seçilmedi");
```

```
            break;
        }
        labelTecRecDoldur();
    }
    private void lblModul1TEC_TextChanged(object sender, EventArgs e)
    {
        if (modul1Durum != 2 && TEC1 <= 127 && REC1 <= 127)
        {
            modul1Durum = 0;
            modul1NameColor = Color.ForestGreen;
            lblModul1Durum.Text = "Hata Aktif";
            lblModul1Durum.ForeColor = Color.ForestGreen;
        }
        if (TEC1 > 127 || REC1 > 127)
        {
            modul1Durum = 1;
            modul1NameColor = Color.Orange;
            lblModul1Durum.Text = "Hata Pasif";
            lblModul1Durum.ForeColor = Color.Orange;
        }
        if (TEC1 > 255)
        {
            modul1Durum = 2;
            modul1NameColor = Color.Red;
            lblModul1Durum.Text = "Bus Off";
            lblModul1Durum.ForeColor = Color.Red;
        }
    }
    private void lblModul1REC_TextChanged(object sender, EventArgs e)
    {
```

```
    if (modul1Durum != 2 && TEC1 <= 127 && REC1 <= 127)
    {
        modul1Durum = 0;
        modul1NameColor = Color.ForestGreen;
        lblModul1Durum.Text = "Hata Aktif";
        lblModul1Durum.ForeColor = Color.ForestGreen;
    }
    if (TEC1 > 127 || REC1 > 127)
    {
        modul1Durum = 1;
        modul1NameColor = Color.Orange;
        lblModul1Durum.Text = "Hata Pasif";
        lblModul1Durum.ForeColor = Color.Orange;
    }
    if (TEC1 > 255)
    {
        modul1Durum = 2;
        modul1NameColor = Color.Red;
        lblModul1Durum.Text = "Bus Off";
        lblModul1Durum.ForeColor = Color.Red;
    }
}
private void lblModul2TEC_TextChanged(object sender, EventArgs e)
{
    if (modul2Durum != 2 && TEC2 <= 127 && REC2 <= 127)
    {
        modul2Durum = 0;
        modul2NameColor = Color.ForestGreen;
        lblModul2Durum.Text = "Hata Aktif";
        lblModul2Durum.ForeColor = Color.ForestGreen;
```

```
    }
    if (TEC2 > 127 || REC2 > 127)
    {
        modul2Durum = 1;
        modul2NameColor = Color.Orange;
        lblModul2Durum.Text = "Hata Pasif";
        lblModul2Durum.ForeColor = Color.Orange;
    }
    if (TEC2 > 255)
    {
        modul2Durum = 2;
        modul2NameColor = Color.Red;
        lblModul2Durum.Text = "Bus Off";
        lblModul2Durum.ForeColor = Color.Red;
    }
}
private void lblModul2REC_TextChanged(object sender, EventArgs e)
{
    if (modul2Durum != 2 && TEC2 <= 127 && REC2 <= 127)
    {
        modul2Durum = 0;
        modul2NameColor = Color.ForestGreen;
        lblModul2Durum.Text = "Hata Aktif";
        lblModul2Durum.ForeColor = Color.ForestGreen;
    }
    if (TEC2 > 127 || REC2 > 127)
    {
        modul2Durum = 1;
        modul2NameColor = Color.Orange;
        lblModul2Durum.Text = "Hata Pasif";
```

```
            lblModul2Durum.ForeColor = Color.Orange;
        }
        if (TEC2 > 255)
        {
            modul2Durum = 2;
            modul2NameColor = Color.Red;
            lblModul2Durum.Text = "Bus Off";
            lblModul2Durum.ForeColor = Color.Red;
        }
    }

    private void lblModul3TEC_TextChanged(object sender, EventArgs e)
    {
        if (modul3Durum != 2 && TEC3 <= 127 && REC3 <= 127)
        {
            modul3Durum = 0;
            modul3NameColor = Color.ForestGreen;
            lblModul3Durum.Text = "Hata Aktif";
            lblModul3Durum.ForeColor = Color.ForestGreen;
        }
        if (TEC3 > 127 || REC3 > 127)
        {
            modul3Durum = 1;
            modul3NameColor = Color.Orange;
            lblModul3Durum.Text = "Hata Pasif";
            lblModul3Durum.ForeColor = Color.Orange;
        }
        if (TEC3 > 255)
        {
            modul3Durum = 2;
```

```
        modul3NameColor = Color.Red;
        lblModul3Durum.Text = "Bus Off";
        lblModul3Durum.ForeColor = Color.Red;
    }
}
private void lblModul3REC_TextChanged(object sender, EventArgs e)
{
    if (modul3Durum != 2 && TEC3 <= 127 && REC3 <= 127)
    {
        modul3Durum = 0;
        modul3NameColor = Color.ForestGreen;
        lblModul3Durum.Text = "Hata Aktif";
        lblModul3Durum.ForeColor = Color.ForestGreen;
    }
    if (TEC3 > 127 || REC3 > 127)
    {
        modul3Durum = 1;
        modul3NameColor = Color.Orange;
        lblModul3Durum.Text = "Hata Pasif";
        lblModul3Durum.ForeColor = Color.Orange;
    }
    if (TEC3 > 255)
    {
        modul3Durum = 2;
        modul3NameColor = Color.Red;
        lblModul3Durum.Text = "Bus Off";
        lblModul3Durum.ForeColor = Color.Red;
    }
}
private void btnTecRecBilgi_Click(object sender, EventArgs e)
```

```
{
   lblTecRecBilgi.Visible = !lblTecRecBilgi.Visible;
}
// Mesaj Önceliklendirme Simülasyonu Butonlar ve Fonksiyonlar
private void
 cmbMesajOnceliklendirmeSenaryoSecimi_SelectedIndexChanged(object
 sender, EventArgs e)
{
   int secilenSenaryo =
cmbMesajOnceliklendirmeSenaryoSecimi.SelectedIndex;
   switch (secilenSenaryo)
   {
      case 0:
         lblSenaryoAciklama.Text = "Birinci senaryo dominant yani 0 olan
 başlangıç biti ile başlamakta ve birinci bitte tüm düğümler resesif olmaktadır.
 Bu durumda herhangi bir önceliklendirme yapılmaz." +
            " İkinci bitte tüm birimler dominant olduğundan yine bir
önceliklendirme gözlemlenmez." +
            " Üçüncü bitte diğer tüm düğümler dominant iken İkinci Modül
 resesif duruma geçer ve önceliğini kaybeder." +
            " Dördüncü bitte tüm düğümler resesif olur ve bir önceliklendirme
 yapılmaz." +
            " Beşinci bitte birinci modül resesif olur ve aynı anda dominant olan
 Üçüncü ve Dördüncü modüller öncelikli olarak kalırlar." +
            " Beşinci bitte birinci modül tekrar dominant olsa bile daha anlamlı
 bitlerde resesif olarak bulunduğundan dolayı burada dominant olması onu geri
 getirmemektedir." +
            " Altıncı bitte hem Üçüncü hem de Dördüncü modüller dominant
 durumdadırlar. Burada bir önceliklendirme yapılmaz." +
            " Yedinci bitte Dördüncü modül resesif Üçüncü modül ise dominant
 olmaktadır. Bu durumda Dördüncü modül de hattan çekilir ve üçüncü modül
 diğerlerinden daha öncelikli olarak hatta mesajını basabilir.";
         break;
      case 1:
```

```
            lblSenaryoAciklama.Text = "İkinci senaryo dominant başlangıç biti ile başlamakta ve birinci bitte tüm düğümler dominant olmaktadır. Bu durumda bir önceliklendirme yapılmaz." +
                " İkinci bitte Dördüncü Modül Resesif olmakta ve diğer düğümlere göre önceliğini kaybetmektedir." +
                " Üçüncü bitte geriye kalan üç modülün hepsi resesif dördüncü bitte ise hepsi dominant olmaktadır. Bu durumda bir önceliklendirme gözlemlenemez." +
                " Beşinci bitte ikinci modül dominant, bir ve üçüncü modüller ise resesif olmaktadır ve bu şekilde önceliklerini kaybetmektedirler." +
                " Bu noktada dominant olması, daha anlamlı bitlerde çekinik kaldığı için dördüncü biti geri getirmemektedir." +
                " Altıncı bit ve sonrasında ikinci modülün diğer modüllere üstün geldiği ve mesajını hatta basabileceği gözlemlenebilir.";
            break;
        default:
            break;
    }
}
private void btnMesajOnceliklendirmeSenaryoBaslat_Click(object sender, EventArgs e)
{
    int secilenSenaryo = cmbMesajOnceliklendirmeSenaryoSecimi.SelectedIndex;
    switch (secilenSenaryo)
    {
        case 0:
            onceliklendirmeTimerCounter1 = 0;
            onceliklendirmeMakeLabelInvisible();
            onceliklendirmeTimer2.Stop();
            onceliklendirmeTimer1.Start();
            break;
        case 1:
```

```
            onceliklendirmeTimerCounter2 = 0;
            onceliklendirmeMakeLabelInvisible();
            onceliklendirmeTimer1.Stop();
            onceliklendirmeTimer2.Start();
            break;
        default:
            onceliklendirmeTimerCounter2 = 0;
            onceliklendirmeMakeLabelInvisible();
            onceliklendirmeTimer1.Stop();
            onceliklendirmeTimer2.Start();
            break;
    }
}
private void onceliklendirmeMakeLabelInvisible()
{
    lblOncelik11.Visible = false;
    lblOncelik12.Visible = false;
    lblOncelik13.Visible = false;
    lblOncelik14.Visible = false;
    lblOncelik21.Visible = false;
    lblOncelik22.Visible = false;
    lblOncelik23.Visible = false;
    lblOncelik24.Visible = false;
    lblOncelik31.Visible = false;
    lblOncelik32.Visible = false;
    lblOncelik33.Visible = false;
    lblOncelik34.Visible = false;
    lblOncelik41.Visible = false;
    lblOncelik42.Visible = false;
    lblOncelik43.Visible = false;
```

```
    lblOncelik44.Visible = false;
    lblOncelik51.Visible = false;
    lblOncelik52.Visible = false;
    lblOncelik53.Visible = false;
    lblOncelik54.Visible = false;
    lblOncelik61.Visible = false;
    lblOncelik62.Visible = false;
    lblOncelik63.Visible = false;
    lblOncelik64.Visible = false;
    lblOncelik71.Visible = false;
    lblOncelik72.Visible = false;
    lblOncelik73.Visible = false;
    lblOncelik74.Visible = false;
    lblOncelik81.Visible = false;
    lblOncelik82.Visible = false;
    lblOncelik83.Visible = false;
    lblOncelik84.Visible = false;
}
// Hat Yükü Hesaplama Butonlar ve Fonksiyonlar
private void btnMesajEkle_Click(object sender, EventArgs e)
{
    lstMessages.Items.Add(txtMessageName.Text + "  -  " +
 cmbCycleTime.SelectedItem.ToString());
    repTimes.Add(Convert.ToInt32(cmbCycleTime.SelectedItem));
    double load = busLoadCalculator(cmbHatHizi.SelectedText);
    lblBusLoad.Text = "%" + load.ToString();
    lgBusLoad.Value = (int)load;
}
private void btnListTemizle_Click(object sender, EventArgs e)
{
```

```
    repTimes.Clear();
    double load = busLoadCalculator(cmbHatHizi.SelectedText);
    lblBusLoad.Text = "%" + load.ToString();
    lgBusLoad.Value = (int)load;
}
private double busLoadCalculator(String busSpeed) {
    int repTime = 0;
    double load = 0;
    for (int i = 0; i < repTimes.Count(); i++)
    {
        repTime = repTimes.ElementAt(i);
        if (busSpeed.Equals("500 kbps"))
        {
            switch (repTime)
            {
                case 10:
                    load += 1.8;
                    break;
                case 50:
                    load += 0.4;
                    break;
                case 100:
                    load += 0.2;
                    break;
                case 500:
                    load += 0.04;
                    break;
                default:
                    break;
            }
```

```
}
else if (busSpeed.Equals("250 kbps"))
{
   switch (repTime)
   {
      case 10:
         load += 3.6;
         break;
      case 50:
         load += 0.8;
         break;
      case 100:
         load += 0.4;
         break;
      case 500:
         load += 0.08;
         break;
      default:
         break;
   }
}
else
{
   switch (repTime)
   {
      case 10:
         load += 5.4;
         break;
      case 50:
         load += 1.6;
```

```
                break;
            case 100:
                load += 0.8;
                break;
            case 500:
                load += 0.16;
                break;
            default:
                break;
        }
      }
   }
   return load;
}
private void btnBusLoadYenile_Click(object sender, EventArgs e)
{
   double load = busLoadCalculator(cmbHatHizi.SelectedText);
   lblBusLoad.Text = "%" + load.ToString();
   lgBusLoad.Value = (int) load;
}
// Real Time Simülasyonu Butonlar ve Fonksiyonlar
private void cmbRealTimeSenaryoSecimi_SelectedIndexChanged(object
 sender, EventArgs e)
{
   int secilenSenaryo = cmbRealTimeSenaryoSecimi.SelectedIndex;
   switch (secilenSenaryo)
   {
      case 0:
         lblRealTimeSenaryolar.Text = "Normal çalışma durumunda araç
 üzerindeki üç adet modül gösterilmektedir. Bunlar Motor Kontrol Modülü,
 Fren Kontrol Modülü ve Gövde Kontrol Modülüdür" +
```

"Bu senaryoda basılan mesajlar motor kontrol modülü tarafından şanzıman durumu, araç hızı ve motor devri bilgileridir. Fren kontrol modülü ise frene basılıp basılmadığı bilgisini" +

"göndermektedir. Gövde Kontrol Modülü bu yapı içerisinde dinleyici konumdadır. Normal senaryoda araç hızı 100 km/saat, motor devri 3500 ve frene basılma durumu basılmadı olarak" +

"gönderilmektedir. Devam eden farklı senaryolarda bu yapılar değişebilir.";

break;

case 1:

lblRealTimeSenaryolar.Text = "Frenleme senaryosu aracın fren yaptığı durumda ilgili tüm mesajların değişimini gözlemlemek amacıyla oluşturulmuştur. Araç fren yaptığı durumda ilk" +

"olarak fren kontrol modülü bunu fark edecek ve frene basılma durumunu basıldı olarak güncelleyecektir. Frene basılması ile birlikte araç hızının ve motor devrinin de düşmesi" +

"beklenmektedir. Araç hızı 100 km/saat hızdan başlayarak 0 km/saat hıza kadar düşüş gösterecektir. Bununla birlikte motor devri de 3500 devirden başlayıp hızla düşecektir." +

"Ancak normal bir araçta olduğu gibi motor devri sıfır değerini görmeyecektir. Bu senaryoda Gövde Kontrol Modülü izleyici konumda olduğundan o modül yardımıyla okunan mesajlar" +

"gözlemlenebilir.";

break;

case 2:

lblRealTimeSenaryolar.Text = "Vites düşürme senaryosunda aracın vites düşürme durumu ve bu durumda diğer mesajların etkilenmesi incelenmiştir. Araç vites düşürdüğünde ilk anda" +

"motor devrinin artması beklenir. Bu durumda ilk önce motor kontrol modülü vites bilgisi mesajını değiştirecek ve bulunan durumdan bir vites geriye çekecektir. Bunun hemen" +

"ardından ise motor devri artış gösterecektir. Frenleme senaryosunun aksine motor devrindeki değişim bu senaryo özelinde daha hızlı olacak ve anlık bir değişim olarak gözlem" +

"lenebilecektir. Aracın frene basılma durumunda ve hızında bir değişim gözlemlenmeyecektir.";

break;

```
        case 3:
            lblRealTimeSenaryolar.Text = "Vites yükseltme senaryosunda aracın vites yükselttiği durumu ve bu durumda diğer mesajların etkilenmesi incelenmiştir. Araç vites yükselttiğinde" +
                "ilk anda motor devrinin azalması beklenir. Bu durumda ilk önce motor kontrol modülü vites bilgisi mesajını değiştirecek ve bulunan vitesten bir üst vitese geçirecektir." +
                "Sonrasında ise motor devrinin azalması gözlemlenecektir. Aynı vites düşürme senaryosunda olduğu gibi burada da motor devrindeki değişim daha anlık olacak ve o şekilde" +
                "gözlemlenecektir.";
            break;
        default:
            break;
    }
}
private void senaryo2Timer_Tick(object sender, EventArgs e)
{
    if (!(hiz == 20 && devir == 15))
    {
        rgHiz.Value = hiz;
        rgDevir.Value = devir;
        hiz = hiz - 4;
        devir = devir - 1;
    }
    else
    {
        senaryo2Timer.Stop();
    }
}
void getAvailableComPorts()
{
```

```
    ports = SerialPort.GetPortNames();
}
private void connectToArduino()
{
    isConnected = true;
    string selectedPort =
 cmbPortSecimi.GetItemText(cmbPortSecimi.SelectedItem);
    port = new SerialPort(selectedPort, 9600, Parity.None, 8, StopBits.One);
    port.Open();
    port.Write("#STAR\n");
    btnBaglan.Text = "Bağlantıyı Kes";
    rgHiz.Value = 100;
    rgDevir.Value = 35;
    lblVitesBilgisi.Text = "4";
    lblFrenDurumu.Text = "Frene Basılmadı";
}
private void disconnectFromArduino()
{
    isConnected = false;
    port.Write("#STOP\n");
    port.Close();
    btnBaglan.Text = "Bağlan";
    rgHiz.Value = 0;
    rgDevir.Value = 0;
    lblVitesBilgisi.Text = "****";
    lblFrenDurumu.Text = "****";
}
private void btnRealTimeGonder_Click(object sender, EventArgs e)
{
    switch (cmbRealTimeSenaryoSecimi.SelectedIndex)
```

```
{
    case 0:
        port.Write("#LED1ON\n");
        rgHiz.Value = 100;
        rgDevir.Value = 35;
        lblVitesBilgisi.Text = "4";
        lblFrenDurumu.Text = "Frene Basılmadı";
        lblFrenDurumu.ForeColor = Color.Maroon;
        break;
    case 1:
        port.Write("#LED2ON\n");
        lblVitesBilgisi.Text = "4";
        lblFrenDurumu.Text = "Frene Basıldı";
        lblFrenDurumu.ForeColor = Color.Green;
        hiz = 100;
        devir = 35;
        senaryo2Timer.Start();
        break;
    case 2:
        port.Write("#LED3ON\n");
        rgHiz.Value = 100;
        rgDevir.Value = 50;
        lblVitesBilgisi.Text = "3";
        lblFrenDurumu.Text = "Frene Basılmadı";
        lblFrenDurumu.ForeColor = Color.Maroon;
        break;
    case 3:
        port.Write("#LED4ON\n");
        rgHiz.Value = 100;
        rgDevir.Value = 25;
```

```
            lblVitesBilgisi.Text = "5";
            lblFrenDurumu.Text = "Frene Basılmadı";
            lblFrenDurumu.ForeColor = Color.Maroon;
            break;
        default:
            break;
    }
}
private void btnBaglan_Click(object sender, EventArgs e)
{
    if (!isConnected)
    {
        connectToArduino();
    }
    else
    {
        disconnectFromArduino();
    }
}
private void btnPortAra_Click(object sender, EventArgs e)
{
    getAvailableComPorts();
    cmbPortSecimi.Items.Clear();

    foreach (string port in ports)
    {
        cmbPortSecimi.Items.Add(port);
        if (ports[0] != null)
        {
            cmbPortSecimi.SelectedItem = ports[0];
```

```
            }
        }
    }
    private void btnRealTimeYenile_Click(object sender, EventArgs e)
    {
        String serialReadData = port.ReadLine();
        lblCanReadMessage.Text = "Fren Modül Mesajı = " + serialReadData;
    }
  }
}
```

Ek-E

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace CANBUSControllerInterface
{
   public partial class form_bilgi : Form
   {
      public form_bilgi()
      {
         InitializeComponent();
      }
      private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
      {
         // Genel Bilgiler Alt Kısımları
         if (treeView1.SelectedNode.Name.Equals("childTarihce"))
         {
            panelTarihçe.Height = 600;
            panelMimari.Height = 0;
            panelHataCercevesi.Height = 0;
            panelTecRec.Height = 0;
            panelAvantajlar.Height = 0;
            panelProtokoller.Height = 0;
```

```
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childMimari"))
{
    panelMimari.Height = 600;
    panelTarihçe.Height = 0;
    panelHataCercevesi.Height = 0;
    panelTecRec.Height = 0;
    panelAvantajlar.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
```

```
else if (treeView1.SelectedNode.Name.Equals("childAvantajlar"))
{
    panelAvantajlar.Height = 600;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childProtokoller"))
{
    panelProtokoller.Height = 600;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
```

```
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
// Çalışma Prensibi Alt Kısımları
else if (treeView1.SelectedNode.Name.Equals("childVeriYolu"))
{
    panelVeriYolu.Height = 600;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childOnceliklendirme"))
{
```

```
    panelMesajOnceliklendirme.Height = 600;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childMultimaster"))
{
    panelMultimaster.Height = 600;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
```

```
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
// CAN Mesaj Yapıları Alt Kısımları
else if (treeView1.SelectedNode.Name.Equals("childCAN20A"))
{
    panelCAN20A.Height = 600;
    panelMultimaster.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childCAN20B"))
{
    panelCAN20B.Height = 600;
    panelCAN20A.Height = 0;
```

```
    panelMultimaster.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
// Bit Süresi ve Hat Yoğunluğu Alt Kısımları
else if (treeView1.SelectedNode.Name.Equals("childKuanta"))
{
    panelKuanta.Height = 600;
    panelCAN20B.Height = 0;
    panelCAN20A.Height = 0;
    panelMultimaster.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
```

```
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childOrneklemeZamani"))
{
    panelOrneklemeZamani.Height = 600;
    panelKuanta.Height = 0;
    panelCAN20B.Height = 0;
    panelCAN20A.Height = 0;
    panelMultimaster.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childBusLoad"))
{
    panelBusLoad.Height = 600;
    panelOrneklemeZamani.Height = 0;
    panelKuanta.Height = 0;
    panelCAN20B.Height = 0;
```

```
    panelCAN20A.Height = 0;
    panelMultimaster.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelVeriYolu.Height = 0;
    panelProtokoller.Height = 0;
    panelAvantajlar.Height = 0;
    panelTecRec.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
}
// CAN Hataları Alt Kısımları
else if (treeView1.SelectedNode.Name.Equals("childErrorFrame"))
{
    panelHataCercevesi.Height = 600;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelTecRec.Height = 0;
    panelAvantajlar.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
```

```
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childTimeOut"))
{
    panelZamanGecikmesi.Height = 600;
    panelHataCercevesi.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelTecRec.Height = 0;
    panelAvantajlar.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childHataOnleme"))
{
    panelHataOnleme.Height = 600;
    panelZamanGecikmesi.Height = 0;
    panelHataCercevesi.Height = 0;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelTecRec.Height = 0;
```

```
    panelAvantajlar.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelBusLoad.Height = 0;
}
else if (treeView1.SelectedNode.Name.Equals("childTECREC"))
{
    panelTecRec.Height = 600;
    panelTarihçe.Height = 0;
    panelMimari.Height = 0;
    panelHataCercevesi.Height = 0;
    panelAvantajlar.Height = 0;
    panelProtokoller.Height = 0;
    panelVeriYolu.Height = 0;
    panelMesajOnceliklendirme.Height = 0;
    panelMultimaster.Height = 0;
    panelCAN20A.Height = 0;
    panelCAN20B.Height = 0;
    panelKuanta.Height = 0;
    panelOrneklemeZamani.Height = 0;
    panelZamanGecikmesi.Height = 0;
    panelHataOnleme.Height = 0;
    panelBusLoad.Height = 0;
}
```

```
        }
        private void btnHataCerceveSimulasyon_Click(object sender, EventArgs e)
        {
            Form3 simulasyonForm = new Form3();
            simulasyonForm.Show();
        }
        private void btnTecRecSimulasyon_Click(object sender, EventArgs e)
        {
            Form3 simulasyonForm = new Form3();
            simulasyonForm.Show();
        }
        private void btnMesajOnceliklendirmeSimulasyon_Click(object sender,
         EventArgs e)
        {
            Form3 simulasyonForm = new Form3();
            simulasyonForm.Show();
        }
        private void btnBusLoadSimulasyon_Click(object sender, EventArgs e)
        {
            Form3 simulasyonForm = new Form3();
            simulasyonForm.Show();
        }
        private void btnEkranAc_Click(object sender, EventArgs e)
        {
            Form3 simulasyonForm = new Form3();
            simulasyonForm.Show();
        }
    }
}
```

KİŞİSEL YAYIN VE ESERLER

[1] **Guclu O.,** Kizir S., Endüstri 4.0 Kapsamında CAN Haberleşmesi Uygulamalarının İncelenmesi, *Bildiriler Kitabı, Elektrik Elektronik Mühendisliği Kongresi,* İstanbul, Türkiye, 16-18 Kasım 2017.

ÖZGEÇMİŞ

1991 yılında İstanbul'da doğdu. İlk, orta ve lise öğrenimini İstanbul'da tamamladı. 2010 yılında girdiği İstanbul Teknik Üniversitesi Kontrol Mühendisliği bölümünü 2014 senesinde bitirdi ve Kontrol Mühendisi olarak mezun oldu. 2010-2014 seneleri arasında birçok yazılım ve haberleşme projesinde görev aldı. 2014 senesinde Kocaeli Üniversitesi Mekatronik Mühendisliği Bölümü'nde başladığı Yüksek Lisans öğrenimine halen devam etmekte ve 2015 senesinden bu yana Ford Otosan firmasında Sistem Mühendisi olarak görevini sürdürmektedir.