**MARMARA ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

# PROGRAMLANABİLİR CİHAZLAR İÇİN KOMUT SETLERİNİ YÖNETME VE KOMUT YANITLARINI İZLEME AMAÇLI ÖZGÜN BİR TEST YAZILIMI GERÇEKLEŞTİRİLMESİ

YASİR KARADENİZ

**YÜKSEK LİSANS TEZİ**

Mekatronik Anabilim Dalı

Mekatronik Programı

**DANIŞMAN**

Prof. Dr. Haluk Küçük

İSTANBUL, 2018

**MARMARA ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

# PROGRAMLANABİLİR CİHAZLAR İÇİN KOMUT SETLERİNİ YÖNETME VE KOMUT YANITLARINI İZLEME AMAÇLI ÖZGÜN BİR TEST YAZILIMI GERÇEKLEŞTİRİLMESİ

YASİR KARADENİZ

(526212011)

**YÜKSEK LİSANS TEZİ**

Mekatronik Anabilim Dalı

Mekatronik Programı

**DANIŞMAN**

Prof. Dr. Haluk Küçük

İSTANBUL, 2018

# MARMARA ÜNİVERSİTESİ
# FEN BİLİMLERİ ENSTİTÜSÜ

Marmara Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Öğrencisi Yasir KARADENİZ'in "Programlanabilir Cihazlar İçin Komut Setlerini Yönetme ve Komut Yanıtlarını İzleme Amaçlı Özgün Bir Test Yazılımı Gerçekleştirilmesi " başlıklı tez çalışması, 20 Haziran 2018 tarihinde savunulmuş ve jüri üyeleri tarafından başarılı bulunmuştur.

**Jüri Üyeleri**

Prof. Dr. Haluk KÜÇÜK (Danışman)

Marmara Üniversitesi .....

Doç. Dr. Hayriye KORKMAZ (Üye)

Marmara Üniversitesi .....

Doç. Dr. Serkan TOPALOĞLU (Üye)

Yeditepe Üniversitesi .....

**ONAY**

Marmara Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..09..07..2018.. tarih ve 2018/16-04 sayılı kararı ile Yasir KARADENİZ'in Mekatronik Anabilim Dalı Mekatronik Programında Yüksek Lisans derecesi alması onanmıştır.

**Fen Bilimleri Enstitüsü Müdürü**

**Prof. Dr. Bülent EKİCİ**

# TEŞEKKÜR

Tez çalışmam süresince desteğini esirgemeyen tez danışmanım sayın Prof. Dr. Haluk Küçük'e, yüksek lisans yapmam konusunda beni teşvik eden ve yüksek lisans sürecinde de desteklerini esirgemeyen arkadaşlarım Dr. Öğr. Üyesi Serkan Aydın, Dr. Barış Doğan ve Dr. Ayşe Yayla'ya teşekkürlerimi borç bilirim.

Ayrıca eksiksiz olarak bütün hocalarıma, yüksek lisans eğitim süreci boyunca her zaman pozitif bir tutum sergilemiş olmalarından dolayı bütün samimiyetimle özel olarak teşekkürü borç bilirim.

# İÇİNDEKİLER

# ÖZET

## PROGRAMLANABİLİR CİHAZLAR İÇİN KOMUT SETLERİNİ YÖNETME VE KOMUT YANITLARINI İZLEME AMAÇLI ÖZGÜN BİR TEST YAZILIMI GERÇEKLEŞTİRİLMESİ

Aygıtların uzak bir noktadan kontrolü artık teknolojinin temel unsurlarından biridir ve nesnelerin interneti gibi kavramlarla gelişimini devam ettirerek, kısa zamanda bütün elektronik aygıtların normal bir özelliği olma yolunda ilerlemektedir.

Uzaktan kontrol ve monitör özelliği olan her cihazın bir komut seti mevcuttur ve bu komut seti cihazın geliştirilmesi esnasında sürekli test edilmek durumundadır. Cihazın geliştirilmesinin yanında, cihaz için uygulama geliştirenler de komut setini test ve analiz etmeye ihtiyaç duyarlar.

Analiz ve test işlemi için ise ayrı bir program gereklidir. Bunun için HyperTerminal, Putty gibi farklı programlar mevcuttur ancak komutların açıklamaları ile birlikte kaydedilmesi, detaylı loglanması, gelen cevabın aynı anda ASCII, HEX ve desimal ve bit seviyesinde görüntülenmesi gibi bazı çok temel özellikler bunların hiçbirinde mevcut değildir. Bu temel özellikler için bir haberleşme programının yanında, farklı yardımcı programlara da ihtiyaç duyulmaktadır.

Bu tez çalışmasındaki temel amaç, bir cihazla haberleşme için gerekli bütün araçları bünyesinde barındıran bir yazılım ortaya çıkarmaktır.

**Mayıs, 2018** **Yasir Karadeniz**

# ABSTRACT

## IMPLEMENTATION OF A NOVEL TEST SOFTWARE FOR MANAGING INSTRUCTION SETS AND MONITORING RESPONSES OF PROGRAMMABLE DEVICES

For any electronic device which has a remote monitoring and control feature via a computer-based system, there is an instruction set and this instruction set should be tested continuously during the development period of the related device. Also, for any kind of control software development for that device, test and analysis of the instruction set is inevitable. For those requirements, a software is needed, which can communicate with devices, send them instructions and receive the reply and then process it. There are many programs which implement those tasks such as Hyperterminal, but they are limited in many respects like saving tested instructions with explanations and logging reply in details. The software that is developed here is not only able to communicate with devices, but also provides the user with some other features which increase development speed and efficiency.

**May, 2018** **Yasir Karadeniz**

# KISALTMALAR

**ASCII** : American Standard Code for Information Interchange

**COM PORT** : Communication Port

**DNS** : Domain Name Server

**DTE** : Data Terminal Equipment

**EIA** : Electronics Industries Association

**GND** : Ground

**HEX** : Hexadecimal

**IP** : Internet Protocol

**IVI** : Interchangeable Virtual Instruments

**LSB** : Least Significant Byte

**MIB** : Management Information Base

**MSB** : Most Significant Byte

**OID** : Object ID

**OSI** : Open Systems Interconnection

**RX** : Receive

**SCPI** : Standard Commands For Programmable Instrumentation

**SNMP** : Simple Network Management Protocol

**SOCK_DGRAM** : Socket Datagram

**SOCK_STREAM** : Socket Stream

**TCP** : Transfer Control Protocol

**TIA** : Telecomunications Industry Association

**TX** : Transmit

**UDP** : Universal Datagram Protocol

**VISA** : Virtual Instrument Software Architecture

# ŞEKİL LİSTESİ

**SAYFA**

# TABLO LİSTESİ

**SAYFA**

# 1. GİRİŞ VE AMAÇ

Bilgisayar, akıllı telefon gibi bir kontrol noktasından çok farklı tipte sistemlerin kontrol edilmesi artık teknolojinin en temel unsurlarından biridir [1-4]. Elektronik sistem geliştirenler, o sisteme SNMP [5,6], SCPI [15], VISA [13,14] gibi veya kendi tanımladıkları ASCII veya binary bir uzaktan kontrol protokolü de eklemek durumundadır.

Bu mekanizma aslında cihaz üzerinde çalışan gömülü yazılıma bir protokol ile haberleşme özelliği ekleme, bir komut kümesi geliştirip, cihazın parametrelerini bu komut kümesi üzerinden görüntüleme ve değiştirme olanağı veren bir özellikler bütünüdür.

Komut kümesine sahip bir cihazla bilgisayar üzerinden haberleşebilmek için ise ayrı bir program gereklidir [13,14] ve bundan dolayı bu amacı yerine getirmeye çalışan çok sayıda değişik özellik ve yapıda program geliştirilmiştir [21-23]. Her programın programlama kısmında ve kullanıcı arayüzünde farklı bir yaklaşım bulunmaktadır. Bu çeşitliliğe rağmen, hala bazı temel eksikliklerin giderildiği pratik yapıda bir program mevcut değildir.

Aşağıda listelenmiş bu eksiklikleri gidermek üzere bir program geliştirmenin gerekliliği gözlemlenmiş ve benzeri programların sağladığı özellikleri de hesaba katarak, onlara alternatif daha gelişmiş bir programın geliştirilmesi kapsamında bu çalışma yapılmıştır.

Maddeler halinde bu nitelikte bir yazılımın geliştirilme ihtiyacını doğuran sebepler sıralanırsa:

- Teknolojinin üstüne kurulduğu en merkezi unsurlardan biri cihazların bir kontrol noktasından yönetilmesidir.
- Cihazların kontrol ve monitör edilmesi gerekli ve bunu için öncelikle cihazların komut kümesinin anlaşılması, test edilmesi gereklidir.
- Cihazla haberleşebilmek ve komut gönderebilmek için program gereklidir.
- Bunu yapan programlar mevcuttur. Ancak cihaz üzerinde çalışma ve geliştirme yapma sürecinde çok ihtiyaç duyulan aşağıda listelenen özellikler bu

programlarda mevcut olmadığından farklı programlar ve yöntemlerle bu açıklar kapatılmak zorunda kalınmaktadır.

  - Her bir cihaz için komut kümesinin kaydedilmesi
  - Komutlar üzerinde byte seviyesinde çalışma yapılması [9,10,18], checkum hesabı [7,8], komuta açıklama yazma, yeni komut ekleme, komutu çoğaltma, komut paketi üretme gibi yardımcı işlemler
  - Gelen byte dizisinin analiz edilmesi, her bir byte için bitlerin direk görüntülenmesi [18]
  - Tarih, gönderilen komut, gelen cevap şeklinde detaylı loglama

- Bu eksiklerin bir laboratuvar ortamında cihazlarla haberleşmek zorunda olan öğrencilerin veya bir projede çalışan geliştirici ekip üyelerinin bir cihazla ilk kez çalışmaya başladıklarında, daha önce başkaları tarafından çözümlenmiş komutlar üzerinde bilimsel kazanım sağlamayan faydasız zorluklar yaşamasına neden olduğu gözlenmiştir
- Aynı zamanda ekip üyelerinden birinin elde ettiği birikimi ekibin diğer üyelerine aktarmasında zorluklar yaşandığı, bilgi birikimi oluşturma ve bilgiyi paylaşma konusunun sistematik bir şekilde ele alınmadığı görülmüştür.
- Daha kısa zamanda halledilecek komut paketini checksum hesaplayarak oluşturma gibi işlerin, checksum hesaplayıcı gibi bazı araçların eksikliğinden dolayı daha uzun zaman aldığı görülmüştür.

Buna göre genel olarak cihazlarla COM PORT arabirimi veya TCP/IP, UDP, SNMP gibi protokoller üzerinden haberleşebilen ve haberleşme ile ilgili parametreleri rahatlıkla değiştirme, kaydetme ve tekrar çağırma özellikleri olan, ASCII veya HEX byte dizisi şeklinde cihaza veri gönderebilen ve ASCII veya HEX olarak görüntüleyen, gönderilen ve alınan paketleri log olarak tarih ve komut açıklaması da olacak şekilde saklayan, gönderilen paketin checksum hesabında yardımcı olan [7,8] desimal, heksadesimal, ASCII dönüşümler ve byte dizisinin desimal bir değere dönüşümü gibi işlemler için [9,10,18] ayrıca hesap makinası açmaya veya başka bir program kullanmaya ihtiyaç duyurmayan, gönderilen komutları her cihaz için ayrı bir dosyada saklayabilen, komutlar hakkında açıklama yazma imkanı veren özelliklere sahip bir

program yazılması gerekir.

Somut bir örnek olarak, bu program ile üniversitenin laboratuvarında bulunan ve bir komut kümesi olan osiloskop, sinyal jeneratörü, voltmetre, robot kol vb. cihazları bilgisayar üzerinden kontrol etmeyi sağlayacaktır. Bunun bir devamı niteliğinde, üzerinde çok sayıda çalışma yapılan internet üzerinden laboratuvar gibi bir uygulamada, orada kullanılacak her bir cihazın bağımsız olarak doğru çalışıp çalışmadığını test etmek için bu programa ihtiyaç vardır. Çünkü bu tip uygulamaları geliştirenler, projenin ilerleyişi esnasında sürekli olarak cihazlara komut gönderme gereği duyarlar ve bunun için kullandıkları programlar birçok açıdan yetersizdir.

Eğitimsel açıdan da bu çalışma, ASCII, HEX, desimal formattaki verileri aynı anda birbirine dönüştürerek üzerinde çalışacak şekilde tasarlandığından bir byte dizisinin ASCI, HEX, desimal formatta yorumlanması gibi programlamanın temel konularını ve TCP, UDP, SNMP, SCPI, VISA, seri haberleşme protokolleri gibi haberleşmenin temel konularının daha hızlı ve kolay kavranmasına yardımcı olur.

Örneğin farklı cihazlarla haberleşerek bunlardan veri okuyan LabView gibi bir program bu haberleşme için cihazı üreten firma tarafından geliştirilmiş sürücüleri kullanılır. Ancak o sürücülerin programla haberleşmesi esnasında alt seviye gerçekleşen olaylar burada gözlenmez.

Ancak bu çalışmada geliştirilen program, o sürücülerin yazılması esnasında kullanılabilecek, yani o sürücüleri yazan kişilerin ihtiyaç duyacağı ve kullanacağı tipte bir program olacaktır.

Haberleşmeyi ve protokolü byte, bit seviyesinde ele alması yönüyle, konuya en alt seviyede yaklaşmaktadır. Bundan dolayı, bu program üzerinden bir cihazla haberleşen kişiler, bir byte dizisinin ne formatta cihaza gönderildiği, checksum'ın buradaki rolünün ne olduğu, checksum'ın nasıl hesaplandığı, gelen byte dizisinin nasıl yorumlandığı gibi konuları kısa zamanda kavrayacaklardır.

Bu çalışmanın amacının en kısa şekilde özeti şudur;

**Üç temel gereklilik:**

- Komut kütüphanesi oluşturma
- Komut paketi oluşturmak için gerekli hesaplamaları otomatize etme

- Cevabın ASCII, HEX, desimal formatlarda ve bit seviyesinde gösterimi

**Üç eğitimsel fayda:**

- Bit, byte, desimal, HEX, ASCII kavramlarının anlaşılması
- Komut protokolü ve checksum kavramlarının anlaşılması
- Temel haberleşmenin anlaşılması (TCP, UDP, SNMP, Seri Haberleşme)

# 2. CİHAZ HABERLEŞMESİNİN TEMEL KAVRAMLARI

## 2.1. Dijital Bilginin İletilmesi

Bir sistemi kontrol etmek için ona bir sinyal göndermek gereklidir. Bu sinyal pnömatik, optik, ultrasonik ve elektriksel olabilir. Bu ortamlar üzerinden analog olarak uzaktaki bir sistemi kontrol etmek mümkündür. Ancak sisteme karmaşık bir bilgi göndererek kontrol etmek gerektiğinde, örneğin sisteme 12.4 gibi bir sayısal değer göndermek gerektiğinde sinyalin sayısallaştırılması gerekir.

Bu durumda sinyal bir kare dalga olacaktır. En alt katmanda, donanım katmanında kare dalga bir sinyal ile bilginin karşı tarafa iletilmesi, en üst katmanda, uygulama katmanında, haberleşmenin sağlanması ile ilgili bazı değerlerin ayarlanması gerekliliğini doğurur.

En çok kullanılan haberleşme teknolojileri, RS-232, RS-422, RS-485, TCP, UDP'dir.

## 2.2. Dijital Bilginin Temel Yapısı

Bit, 1 veya 0 değerinde olan en küçük dijital bilgi birimidir. 4 bit bir nible, 8 bit bir byte olarak adlandırılır.

**Tablo 2.1** Farklı Sayı Sistemlerinde Verinin Gösterimi

| İkilik Sistem | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | nible | | | | nible | | | |
| | byte | | | | | | | |
| HEX | 0xA | | | | 0x6 | | | |
| Ondalık Sistem | 10 | | | | 6 | | | |

Standart olarak bir sayısal değişkenin önüne 0x konularak onun HEX olduğu belirtilmiş olur. Dolayısı ile C, C++, C# gibi programlama dillerinde a=0x10 şeklinde

değişkene bir değer atandığında, desimal olarak 16 değeri atanmış olur. Ancak kimi durumlarda, HEX varsayılan formattır. Örneğin Python dilinde "**bytearray.fromhex('0A 10 0B')"** şeklinde, heksadesimal değerler önünde 0x kullanmaya gerek olmadan bir dizi oluşturulabilir.

### 2.3. Endianness

Endianness bir byte dizisinin en önemli değerinin (MSB) belleğin hangi tarafından başlayacağı ile ilgili bir kavramdır. Little-Endian ve Big-Endian olmak üzere iki tip endianness vardır.

**Tablo 2.2** Byte dizisi olarak ifade edilen bir değerin bellekteki duruşu

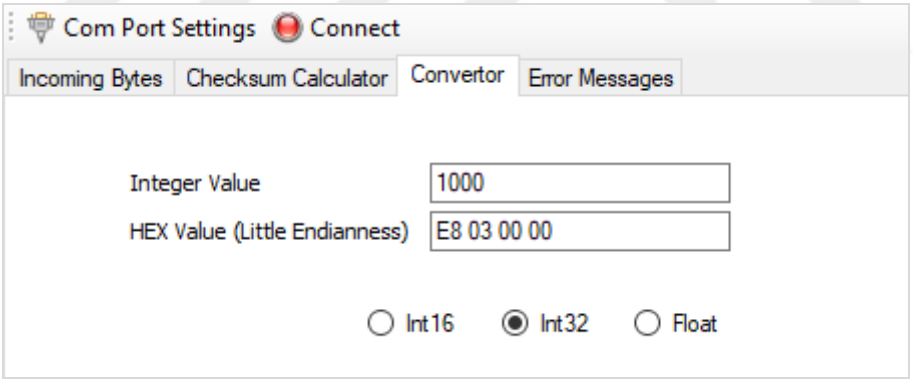
| **Big Endian** | | **Little Endian** | |
|---|---|---|---|
| **BELLEKTEKİ KONUM** | **DEĞER** | **BELLEKTEKİ KONUM** | **DEĞER** |
| Taban Adres + 0 | 00 | Taban Adres + 0 | 01 |
| Taban Adres + 1 | 00 | Taban Adres + 1 | 00 |
| Taban Adres + 2 | 00 | Taban Adres + 2 | 00 |
| Taban Adres + 3 | 01 | Taban Adres + 3 | 00 |

Big-Endian sistemde byte dizisi adrese en önemli değerden (MSB), Little-Endian sistemde ise byte dizisi adrese en önemsiz değerden (LSB) başlayarak yerleştirilir. Örneğin 1 sayısı hafızada Big-Endian sistemde 00 00 00 01 şeklinde, Little-Endian sistemde 01 00 00 00 şeklinde durur.

**Tablo 2.3** Mikroişlemciler ve endianness

| | |
|---|---|
| Intel x86 | Little Endian |
| ARM | Little Endian |
| Power PC | Big Endian |
| Motorola 68000 | Big Endian |
| TCP Network Protocol | Big Endian |

Üst seviye uygulamalarda pratik açıdan etkisi olmayan bu bilginin alt seviyede dikkate alınması zorunludur ve bazı haberleşme protokollerinin bu bilgi olmaksızın anlaşılması mümkün değildir. Bundan dolayı, bu çalışmada geliştirilen uygulamada desimalden heksadesimale x86 mimarisinin kullandığı little-endianness referans alan bir dönüştürücü mevcuttur.




**Şekil 2.1** Program üzerinde HEX dönüştürücü

Yukarıdaki örnekte 1000 sayısının hafızadaki görünümü E8 03 olarak hesaplanmıştır. 0xE8*1 + 0x03*256 = 232+ 762 = 1000

## 2.4. Checksum

Bir byte dizisinin doğru iletilip iletilmediğinin kontrol edilebilmesi için byte dizisinin sonuna mevcut byte dizisi üzerinde yapılan bir hesaplamanın sonucu olarak elde edilen bir veya birden fazla byte eklenir. Bu eklenen byte veya byte dizisine

checksum denir.

Farklı haberleşme protokollerinde farklı checksum hesaplama yöntemleri kullanılmaktadır.

Örnek olarak RS-232 haberleşmede yaygın olarak XOR checksum kullanılır. XOR checksum hesabı, byte dizisindeki bütün elemanların birbirine XOR işlemi uygulanması şeklinde yapılır. Yani "A B C" şeklinde bir byte dizisinde checksum = A XOR B XOR C dir.

Şekil 2.2'de aynı zamanda DeviceTester uygulamasında da kullanılan Python fonksiyonu verilen bir HEX byte dizisinin XOR checksum değerini döndürür.

```
def getCheckSum(packet):
    byteArray = bytearray.fromhex(packet)
    checksum = 0
    for single_byte in byteArray:
        checksum ^= single_byte
    return '{:02X}'.format(checksum)
```

**Şekil 2.2** XOR checksum döndüren Python fonksiyonu

## 2.5. Haberleşme Protokolleri

Haberleşme için kullanılan byte dizisinin yapısını yazılımsal olarak tanımlayan algoritmaya protokol denir.

En basit protokol örneği komutların ASCII olarak gönderilmesidir. Bu protokolde komutlar ASCII birer metinden ibarettir. Örnek olarak "STATUS" komutu bir cihazın mevcut durumuyla ilgili bilginin alınmasını sağlar.

RS-232 ve RS-485 haberleşmede sık kullanılan bir protokol ise şu şekildedir.

| STX (0x02) | ADRES | KOMUT | ETX(0x03) | XOR Checksum |
|---|---|---|---|---|

**Şekil 2.3** STX-ETX protokolü

Şekil 2.3'te görüldüğü gibi komut paketinin başlangıç (STX) ve bitişinde (ETX) 02 ve 03 özel karakterleri kullanılır. Byte dizisinin son kısmına XOR checksum eklenir.

Örnek olarak cihaz adresi heksadesimal olarak 30, komut heksadesimal olarak 31 ise komut paketi "02 30 31 03 00" şeklinde olacaktır.

Benzer bir başka protokol bu protokolün ASCII versiyonu olarak geçer.

| | | | | |
|---|---|---|---|---|
| { | ADRES | KOMUT | } | Mod95 Checksum |

**Şekil 2.4** STX-ETX ASCII protokol

Bu komut protokolleri ile çalışanlar sürekli bir paketin checksum değerini hesaplamak durumundadır. Bu çalışmada geliştirilen uygulamada ise komut paketi checksum kısmına kadar yazıldığında, otomatik olarak checksum değerleri görüntülenir ve bu şekilde başka bir yazılıma ihtiyaç kalmadan checksum hesaplanır.

## 2.6. Çok Kullanılan Haberleşme Teknolojileri

En çok kullanılan iki haberleşme teknolojisi seri port üzerinden ve ethernet üzerinden haberleşmedir.

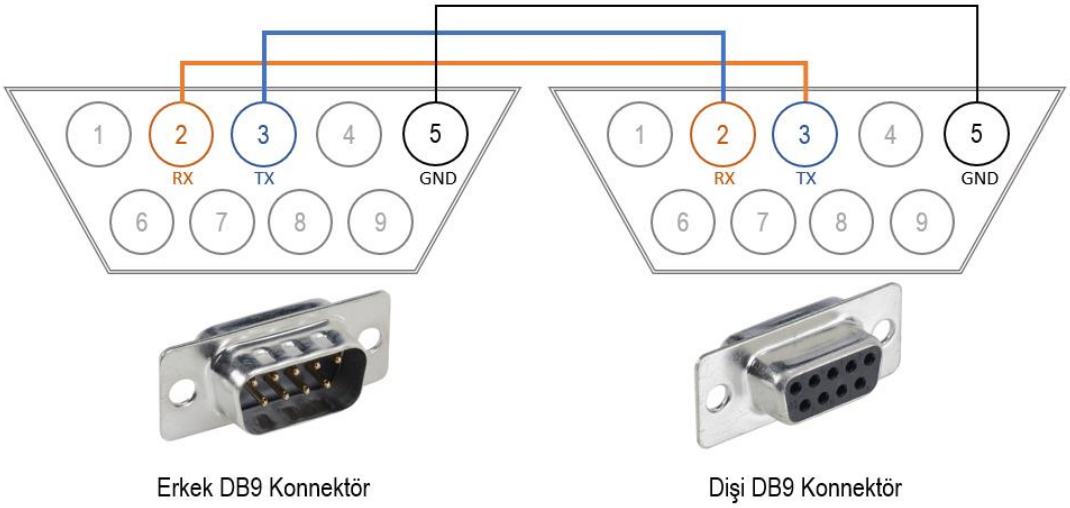
### 2.6.1. RS-232 Seri Haberleşme Protokolü

1962'de ilk versiyonu geliştirilmiştir. RS-232C, EIA RS-232 veya kısaca RS-232 şeklinde kullanılır ve standardı 1969 yılında Electronics Industries Association tarafından belirlenmiştir. O tarihten sonra sürekli farklı ve daha gelişmiş teknolojiler ortaya çıkmasına rağmen, halen cihaz haberleşmesiyle ilgili bir çalışmada incelenmesini zorunlu kılacak kadar yaygındır.

Donanımsal açıdan ele alındığında, RS-232, 1 ve 0 bilgisini voltaj seviyeleri biçiminde iletir. -12V ve -3V arası 0, +3V ve +12V arası 1 olarak tanımlıdır. -3V ve +3V arası sinyal gürültüsüne karşı önlem olarak ölü bölge olarak tanımlanır.

RS-232 için RJ11, RJ45 (EIA-TIA 561 standardı), terminal gibi farklı konnektör tipleri kullanılabilir. Ancak en sık kullanılan ve RS-232 ile özdeşleşmiş konnektör, DB9 konnektördür.

En temel kullanımda, bir alıcı (receive, RX), bir iletici (transmit, TX) ve bir toprak (ground, GND) pini bulunur ve 2 RX, 3 TX, 5 GND'dir.



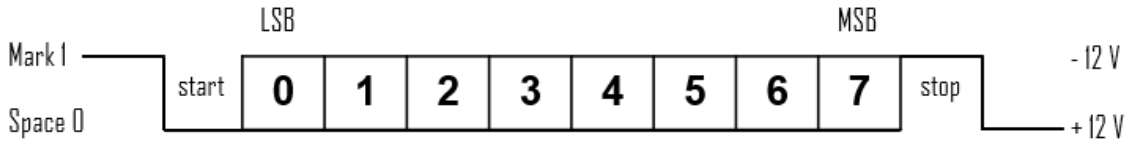

**Şekil 2.5** RS-232 DB9 konnektör ve bağlantı şekli

Genel olarak kontrol ve monitör işleminin yapıldığı cihaz DTE (Data Terminal Equipment), kontrol edilen cihaz DCE (Data Communication Equipment) olarak adlandırılır. Yaygın kullanımda DTE için erkek konnektör tercih edilir.

Dizüstü ve masaüstü bilgisayarlarda RS-232 DB9 konnektör şeklinde COM PORT büyük ölçüde terkedilmiş olmakla birlikte halen bazı anakartlarda ve geliştirme kitlerinde bu konnektör bulunabilmektedir.

Üzerinde bu tip konnektörün olmadığı bir bilgisayar ile RS-232 haberleşmenin gerekli olduğu durumlarda çözüm, "USB to RS-232" dönüştürü kullanılmaktır.

Şekil 2.6'da RS-232 seri haberleşme yapısı görülmektedir. Sayısal veri tek bir pin üzerinden sıralı olarak gönderilir. 8 bit veri, 1 bit stop veya 7 bit veri bir bit stop şeklinde seçenekler mevcuttur. Bu durum uygulamanın en üst katmanına yansır ve RS-232 haberleşme sağlayacak yazılım bu ayarları içermek durumundadır.




**Şekil 2.6** RS-232 sayısal bilgisi

RS-232 seri haberleşme için gerekli ayar parametreleri ve açıklamaları Tablo 2.4'te verilmiştir.

**Tablo 2.4 Seri haberleşme parametreleri**

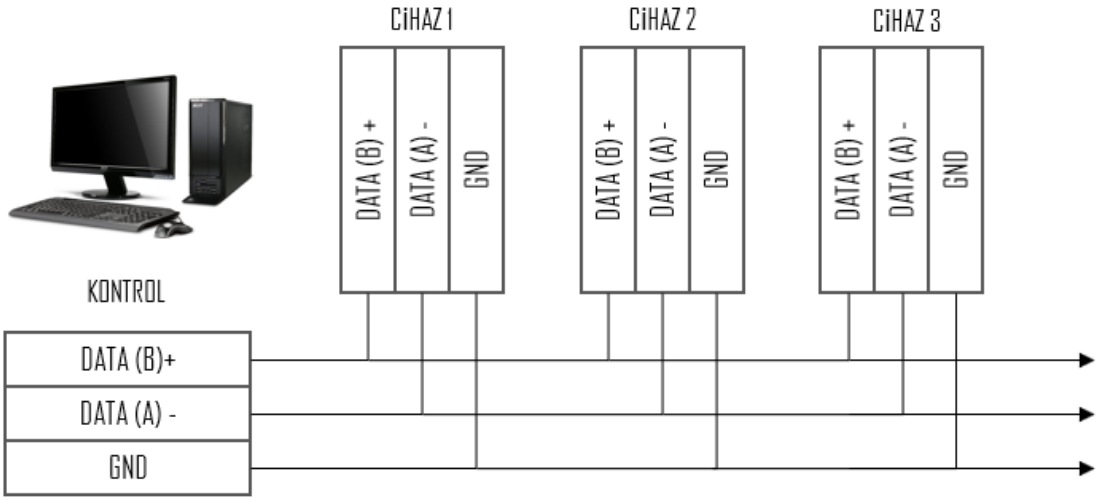
| | |
|---|---|
| Baud Rate | Saniyedeki sinyal değişimi ölçüsüdür. Alıcı ve vericinin baud rate değeri aynı olmak zorundadır. |
| Parity | Paketin doğruluğunu kontrol amaçlı bir bittir. None, odd, even seçenekleri mevcuttur. |
| Data Bits | Bilginin bit adedini belirler. 7 ve 8 seçenekleri mevcuttur. |
| Stop Bit | Bilginin bitiş noktasının bit adedini belirler. 1, 1.5, 2 olabilir. |

Uygulama katmanında alıcı ve verici ayarlarının aynı olması yeterlidir ve haberleşmenin donanımsal yönünün detaylı bilinmesi gerekli değildir. Ancak sinyal gürültüsü, kablo uzunluğundan kaynaklı sinyal zayıflaması, topraklama gibi bazı problemler, üst katmanda etkisini haberleşmenin gerçekleşmemesi veya anlamsız veri gelmesi şeklinde göstermektedir ve bu kullanıcıların problemlerin yazılımdan mı, donanımdan mı kaynaklandığına karar verebilmesi için haberleşmenin donanımsal yapısı hakkında da bilgi sahibi olmalarını gerektirmektedir.

### 2.6.2. RS-485 Seri Haberleşme Protokolü

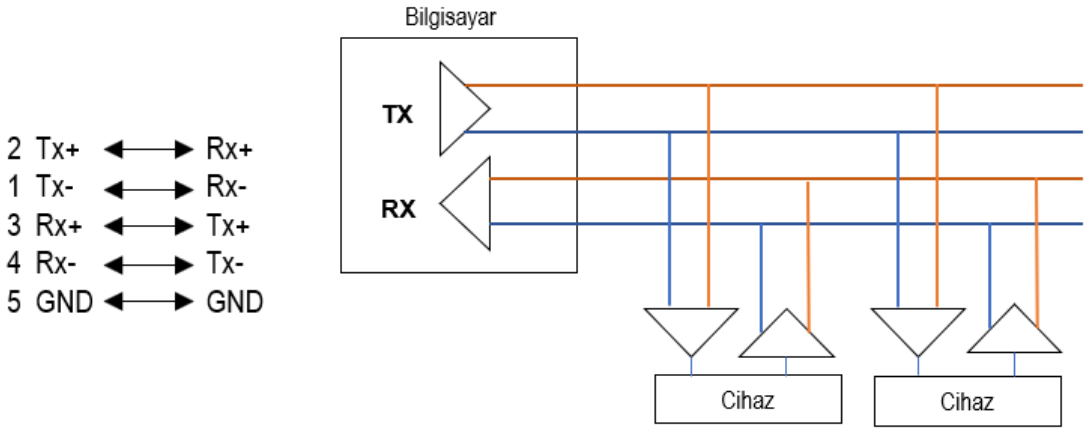
RS-232 kısa mesafeli ve düşük hızlı bir iletişim teknolojisidir. Ayrıca sadece iki ünite arasında haberleşmeye olanak sağlar.

Bu eksiklikler TIA-485 olarak da bilinen RS-485 seri haberleşmenin geliştirilmesini gerektirmiştir. RS-485 seri haberleşme protokolü uzak mesafelerde ve tek bir zincirde Şekil 2.7'de gösterildiği gibi birden fazla cihazın haberleşmesini mümkün kılar.




**Şekil 2.7** RS-485 ile birden fazla ünitenin kullanımı

Sinyal iletiminde gürültü farksal bir devre ile minimize edilmiştir. Bunun için RX+, TX+, RX-, TX- pinleri kullanılmaktadır.

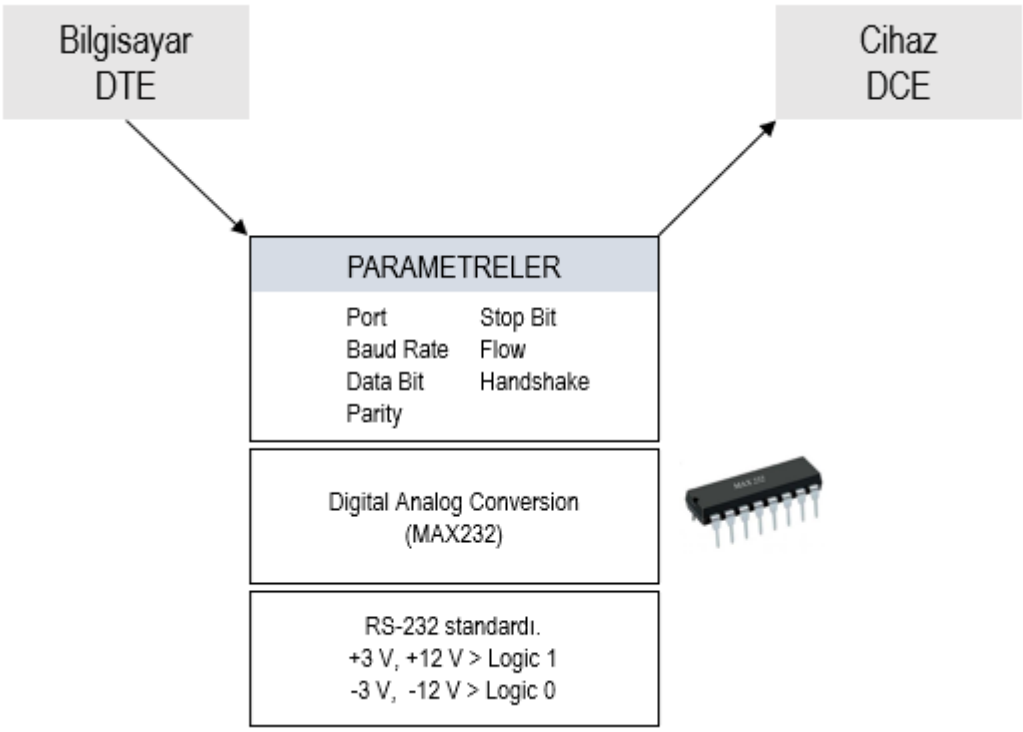



**Şekil 2.8** RS-485 full-dublex bağlantı yapısı

Şekil 2.8'de RS-485 seri haberleşme için genel bağlantı yapısı gösterilmiştir. Ancak üreticiler farklı bağlantı tipleri ve bunun yansıması olarak farklı bağlantı yapıları kullanmaktadır. Üzerinde çalışılan cihazın dökümantasyonunda bağlantı yapısı belirtilir.

RS-485 seri haberleşmenin kullanıldığı bir sistemde, zincire ekli her ünitenin bir adresi mevcuttur. Bu adres, protokoller kısmındaki örnek protokollerde bir byte ile ifade edilen adrestir.

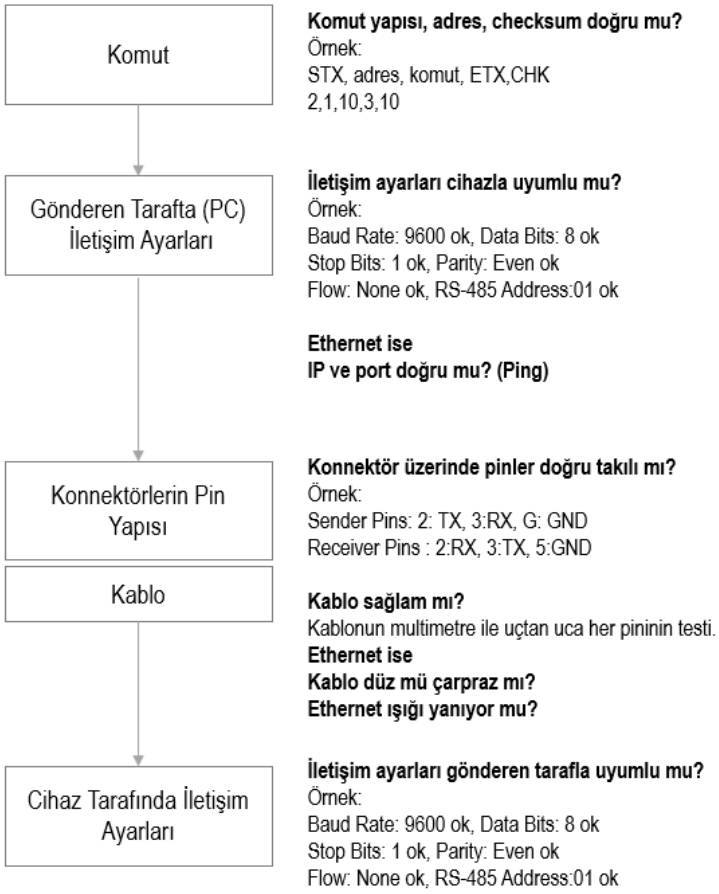
RS-485 sıklıkla RS-232 seri haberleşmede kullanılan DB9 konnektörü kullanır.




**Şekil 2.9** Seri haberleşmenin tüm katmanları

Şekil 2.9'da uygulama katmanından itibaren seri haberleşmeyi oluşturan ve etkileyen katmanlar gösterilmiştir. Cihaz kontrol ve monitor uygulamalarında ideal durum, en üst blokta haberleşme ile ilgili parametrelerin doğru bir şekilde ayarlanması sonucu haberleşmenin gerçekleşmesidir. Böyle bir durumda, fiziksel bağlantının yapılması dışında alt bloklarla bilgisel anlamda yüzleşme gerekliliği olmaz.

Ancak sıklıkla haberleşme sürecinde problemler yaşanmaktadır. Bu problemlerin hızlı bir şekilde çözülebilmesi için Şekil 2.10'da gösterilen adımlardan oluşan bir zincir mevcuttur.




**Şekil 2.10** Bağlantı kontrol şeması

Bu zincirin başından sonuna şu soruların sorulması gereklidir:

- Komut protokolü doğru uygulandı mı? Yani cihaz adresi doğru girildi mi? Checksum doğru hesaplandı mı?
- Cihaz ile bilgisayar arasındaki kablonun bağlantıları doğru mu? Kablo sağlam mı?
- Bilgisayarın RX ve TX portları loop yapıldığında, uygulama üzerinden sorgulama yapıldığında cevap alınıyor mu?
- COM PORT parametreleri gönderen ve alıcı tarafta doğru ayarlandı mı?

Kullanıcılar yoğunlukla Şekil 2.9'da gösterilen en üst katmanla yüzleşirler ve bu yüzleşme seri haberleşme ile bir cihazı en temel seviyede kontrol etmek için yeterli olan aşağıda gösterilen Python kodundan ibarettir.

```
import serial
ser = serial.Serial('COM3', 9600, timeout=0)

command = '10 05 55 AA 01 01 2C 3F'
ser.write(bytearray.fromhex(command))
incomingdata = ser.readline()

#ASCII formatta göster
print (incomingdata)
#HEX formatta göster
print(" ".join("{:02x}".format(c) for c in incomingdata))

ser.close()
```

**Şekil 2.11** Python ile temel seri haberleşme

Şekil 2.11'deki kodun ikinci satırında port ve baud rate ayarlanmıştır. Seri haberleşme ile ilgili diğer ayarlar Python dili için varsayılan değerler olup, Python, 8 bit veri, 1 bit parite şeklinde en çok kullanılan ayarları varsayılan ayar olarak kabul eder.

### 2.6.3. Transmission Control Protocol/Internet Protocol (TCP/IP)

1970'lerde geliştirilen internetin temel haberleşme protokolü olan TCP/IP, endüstriyel sistemlerin kontrol ve monitöründeki haberleşme işinde de gün geçtikçe daha yaygın şekilde kullanılmaktadır.

Özellikle son yıllarda, artık cihazlar için TCP/IP desteği kaçınılmaz hale gelmiştir. Sinyalin gürültüsüz bir şekilde gerektiğinde birden fazla cihaza ulaştırılması görevini TCP/IP, cihazı bir ağa dahil ederek farklı uzak noktalardan erişme imkânı vermek gibi önemli bir özelliği de sunarak icra etmektedir.

Bu durum yeni üretilen cihazlarda TCP/IP desteği olması, hatta eski cihazların da TCP/IP kullanabilmesi için "RS-232 to Ethernet", "RS-485 to Ethernet" şeklinde dönüştürücüler geliştirilmesi sonuçlarını doğurmuştur.

Ayrıca kolaylıkla TCP/IP üzerinden bir sistemi kablosuz bağlantı üzerinden kontrol imkânı veren ESP-8266 gibi modüller geliştirilmiştir.

Teknolojideki bu gelişmeler IoT (Internet of Things) sonucunu doğurmuştur.

TCP/IP haberleşmenin uygulama katmanında kullanıcıya yansıyan kısmı IP ve Port parametreleridir. Bir cihaza bağlanıp kontrol ve monitör etme işlemi için o cihazın IP ve Port değerleri bilinmelidir.

TCP/IP ile haberleşme esnasında bazı güçlükler çıkmaktadır. Sıklıkla belirli bir IP

adresine sahip cihazın ağ üzerinde var olup olmadığının test edilmesi gerekir. Bunun için tek çözüm komut konsolu üzerinden ping komutu kullanmaktır. İkinci problem, bir ağa bağlı tüm aygıtların IP adreslerini görmenin gerekliliğidir. Program üzerinde bunu da sağlayan bir özellik mevcuttur. Üçüncü gereklilik, çalışılan aygıtın IP ve port değerlerinin kaydedilmesidir.

Bu tezde geliştirilen uygulamada bu tip çalışmalar esnasında gözlemlenen bu gereklilikler için çözümler üretilmiştir.

```
import time
from socket import *

#SOCK_STREAM > TCP, SOCK_DGRAM > UDP
clientSocket = socket(AF_INET, SOCK_STREAM)

ip = "127.0.0.1"
port = 12000
addr = (ip, port)
#TCP protokolünde bağlantı gereklidir.
clientSocket.connect(addr)
message = 'test'.encode()
clientSocket.sendall(message)
#Gönderme işleminden sonra bağlantı kapatılıyor.
clientSocket.close()
```

**Şekil 2.12** TCP bağlantı sağlayan istemci Python kodu

Şekil 2.12'deki Python kodu 127.0.0.1 IP ve 12000 port değeri üzerinden bir sunucuya bağlanıp, 'test' şeklinde bir mesaj göndermektedir. Kod üzerinde TCP'nin yapısının yansıması olarak dikkat edilecek noktalar, bağlantı şeklinin TCP olmasını sağlamak üzere "socket" nesnesinin SOCK_STREAM parametresi ile oluşturulması, bağlantının "connect" fonksiyonu ile bağlanması ve işlem bittikten sonra "close" fonksiyonu ile bağlantının kapatılmasıdır.

Bağlantının kapatılmaması durumunda, aynı kod tekrar çalıştırıldığında sunucuya bağlanmada sorun çıkacaktır. Bu durum programda her komut gönderildiğinde bağlantı yapılıp işlem bitince bağlantının kesilmesi ve bağlantının başlangıçta bir kez yapılıp, kullanıcı "bağlantıyı kapat" komutu gönderene kadar bağlı olarak çalışılması şeklinde iki farklı seçeneği göz önünde bulundurmayı gerektirmiştir.

Örneğin kullanıcı bir cihazla çalışırken, HyperTerminal gibi bir program üzerinden cihaza bağlanmış ise başka bir programla cihaza bağlanıp komut gönderemez. Bağlantı

TCP yapısı gereği cihaz tarafından reddedilir. Ancak bu tezde geliştirilen program ile cihaz üzerinde çalışma yapanlar, programın bağlantı seçenekleri arasında bulunan TCPIPStandard bağlantısız çalışma seçeneğini kullanarak başka bir program üzerinden de cihaz komut göndermeye devam edebilirler.

### 2.6.4. User Datagram Protocol (UDP)

Bağlı (Connection oriented) ve doğrulamalı (Acknowledged) bir yapı üzerine kurulu ve bundan dolayı bir gecikme ile çalışan TCP/IP'ye alternatif olarak, verinin karşı tarafa ulaşıp ulaşmadığının doğrulanmadığı ve bağlantı kurmaya gerek olmadan ve bunların sonucu olarak daha hızlı çalışan UDP geliştirilmiştir.

İnternet üzerinde DNS ve SNMP gibi uygulamalar UDP kullanır. Öte yandan cihaz kontrolünde UDP'nin kullanım oranı TCP'den düşüktür.

```
import time
from socket import *

clientSocket = socket(AF_INET, SOCK_DGRAM)

ip = "127.0.0.1"
port = 12000
addr = (ip, port)
message = 'test'.encode()
clientSocket.sendto(message, addr)
data, server = clientSocket.recvfrom(1024)
```

**Şekil 2.13** UDP bağlantı sağlayan istemci Python kodu

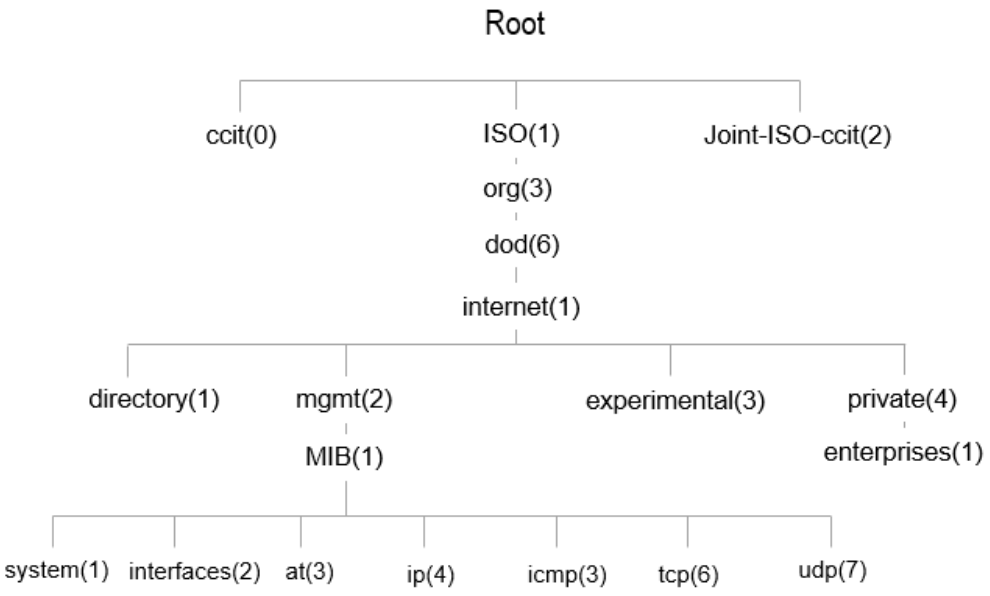
Şekil 2.13'teki Python kodu, mevcut bilgisayar üzerinde 12000 portuna UDP üzerinden bağlantı yapıp "test" şeklinde bir mesaj göndermektedir. Sunucudan gelen cevap 1024 numaralı porttan alınmaktadır.

Bu kod üzerinde dikkat edilmesi gereken iki nokta vardır. Birincisi SOCK_DGRAM parametresidir. Bu socket nesnesinin UDP'ye göre oluşturulmasını sağlar. İkinci nokta ise, TCP örneğindeki "connect" ve "close" fonksiyonlarının bu kodda yer almamasıdır. Çünkü UDP bağlantısız çalışan bir protokoldür. Bunun koda yansıması bu şekilde görülmektedir.

### 2.6.5. Simple Network Management Protocol (SNMP)

1990'larda geliştirilen SNMP (Simple Network Management Protocol) UDP üzerine kurulan bir haberleşme protokolüdür. Bir ağa bağlı cihazların monitör ve kontrol edilmesi için geliştirilmiştir.

SNMP'nin V1, V2, V3 şeklinde üç versiyonu mevcuttur. V3 güvenlik amacıyla geliştirilmiş olup, bu versiyonda veri paketleri şifrelenerek gönderilmekte ve kimlik doğrulaması yapılmaktadır.




**Şekil 2.14** SNMP yapısı

Şekil 2.14'de görüldüğü gibi, SNMP ağaç yapısı şeklinde bir komut protokolüdür. Her bir cihazın komutları ağaç şeklinde bir yapı içeren MIB (Management Information Base) dosyası adı verilen dosyalarda saklıdır.

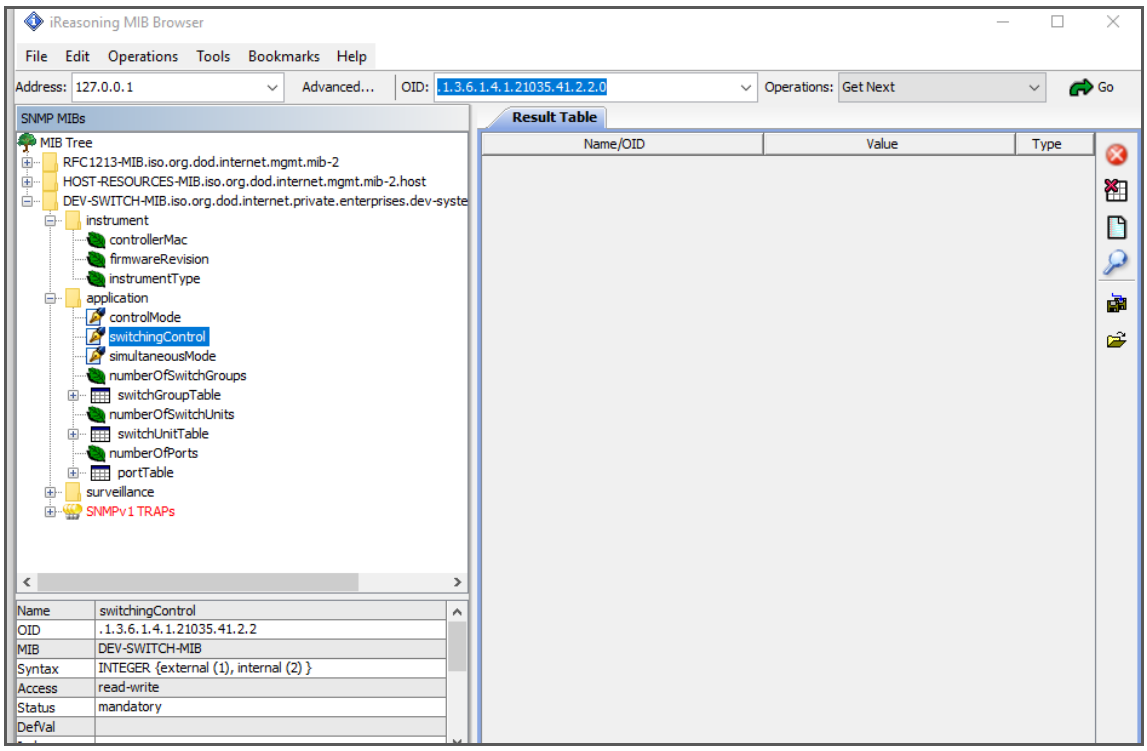
Cihazla ilgili her bir parametre bu ağaçta yer alan bir "node" dur. Buna OID (Object ID) denir. Bu OID'ler ile monitor ve kontrol işlemi yapılır.

Örneğin 1.3.6.1.4.1.21035.41.2.2.0, cihazla ilgili bir parametrenin değerini almak üzere cihaza gönderilecek komut durumundadır. Bu OID yukarıdaki şema referans alınarak analiz edildiğinde, 1.3.6.1.4.1 ile ISO > ORG > DOD > INTERNET > PRIVATE > ENTEPRISES noktasına ulaşılır. Bundan sonraki değer her bir kurum için

verilen özel bir değerdir ve SNMP organizasyonundan alınmak zorundadır.

Bu listede kayıtlı tüm firmalara https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers adresinden ulaşılabilir. Listede mevcut olan Marmara Üniversitesi'nin kodu 39187'dir. Bu Marmara Üniversitesinde SNMP protokolü kullanılarak geliştirilecek bir sistemin komutlarının tamamının 1.3.6.1.4.1.39718 ile başlayacağını gösterir.

Örnek olarak burada 21035 menşei Almanya olan bir firmanın kodudur. Firma bu kodu aldıktan sonra, ağacın geri kalan kısmını kendi ihtiyaçlarına göre düzenler.




**Şekil 2.15** Bir cihazın MIB dosyasının görünümü

SNMP için kullanılmak üzere geliştirilmiş programlar mevcuttur. Şekil 2.15'de görülen iReasoning MIB Browser bunlardan biridir. Bu şekilde sol kısımda bir parametrenin OID değeri ve sol alt tarafta açıklaması görülmektedir.

### 2.6.6. Virtual Instrument Software Architecture (VISA) ve Standard Commands for Programmable Instrumentation (SCPI)

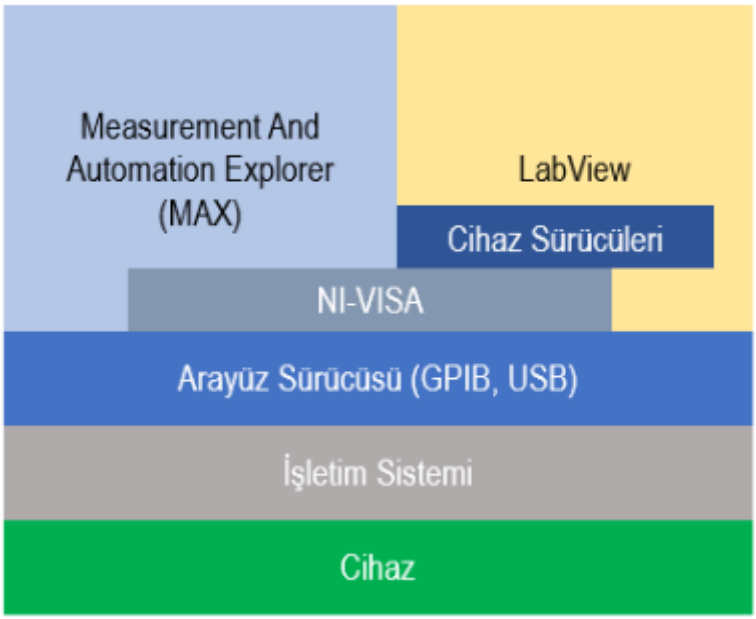
Ölçüm cihazları geliştiren firmalar ve bunlar için kontrol yazılımları geliştiren firma ve kişiler, her firmanın cihazında temelde aynı işlemler için farklı tipte komut küme ve

yapıları geliştirmenin zorluğunu görerek çözümler geliştirmişlerdir.

Program geliştirme maliyetlerini düşürmek, performansı arttırmak ve farklı cihazlar arası uyumu sağlamak üzere 1998'de oluşmaya başlayan Interchangeable Virtual Instruments (IVI) Foundation 2001'de resmileşti.

IVI Foundation cihaz kontrol komutları için spesifikasyon tanımı yapan bir konsorsiyumdur. Üyeleri cihaz ve yazılım üreten firmalar, sistem entegratörleri ve son kullanıcılardan oluşmaktadır.

Bu konsorsiyuma aynı amaç için çalışan iki farklı konsorsiyum daha eklenmiştir. Bunlardan biri VXI Plug&Play System Alliance ve diğeri The Standard Commands for Programmable Instrumentation (SCPI)'dır [25].




**Şekil 2.16** Cihaz ve uygulama katmanı arasında VISA'nın konumu

Şekil 2.16'da VISA'nın cihazdan başlayarak en üst katmandaki uygulamaya kadar giden zincirde konumu görülmektedir. Cihazla haberleşebilmek için öncelikle kullanılan bilgisayar sisteminde donanımsal arabirim mevcut olmalıdır. Bu ethernet, GPIB, USB olabilir. Ayrıca donanım sürücüleri işletim sistemi üzerinde kurulu olmalıdır.

Bu aşamalardan sonra, VISA katmanı mevcuttur. VISA kullanarak haberleşme sağlayabilmek için VISA sürücüleri IVI Foundation sitesinden indirilmelidir [25]. Veya National Instruments sitesinden de NI-VISA sürücüleri indirilebilir [26]. National Instruments firmasının sağladığı bu paket VISA sürücüleri dışında "Measurement And

Automation Explorer" uygulamasını da barındırır. Bu uygulama sistemde mevcut cihazların tespit edilmesi, onlara test komutları gönderilmesi gibi işlemler yapan bir destek uygulamasıdır. Örneğin bu uygulama üzerinde bir cihaz görüntülenemiyorsa cihazın sistem sürücüleri ile ilgili problem var demektir ve bu tezde geliştirilen uygulama ile de cihazla haberleşilemez.

Bu aşamadan sonra örnek olarak PyVisa eklentisi Windows komut konsolu üzerinden "C:\>pip install pyvisa" komutu kullanılarak kurulduğunda, Python üzerinden cihaz ile haberleşmek mümkündür.

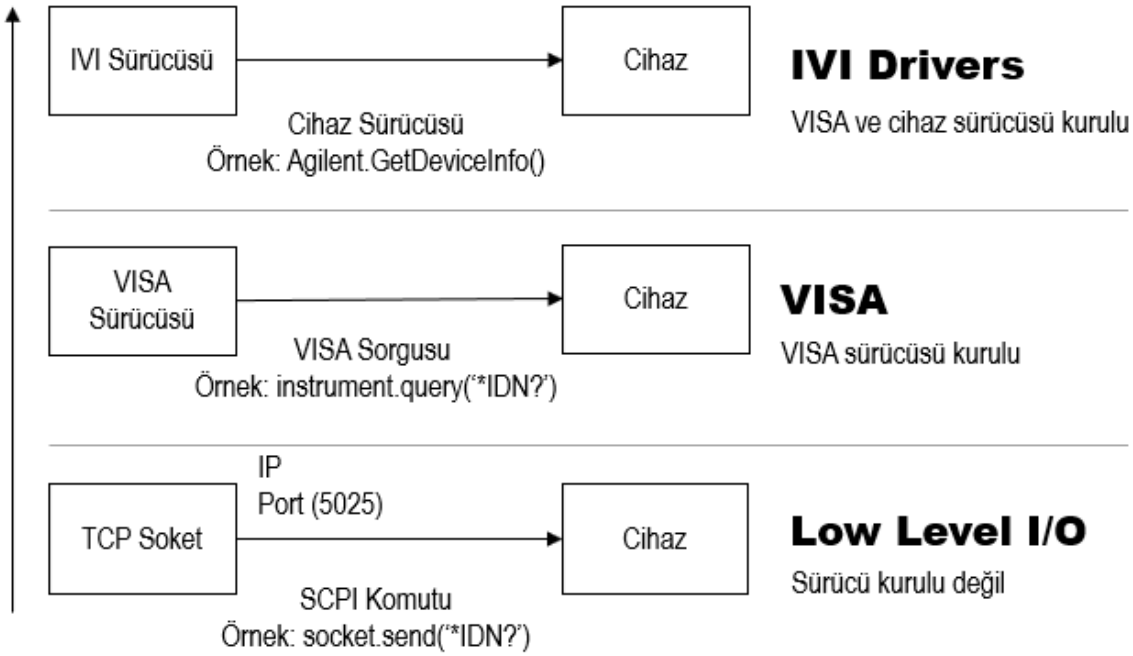
```
import sys
import visa

def sendCommand(scpiCommand):
    try:
        rm = visa.ResourceManager()
        a = rm.list_resources()
        if(len(a)>0):
            inst = rm.open_resource(a[0])
            return inst.query(scpiCommand)
        else:
            return "no resource"

    except Exception as e:
        print(e)
    finally:
        try:

        except Exception:
            pass

def test():
    print (sendCommand('*IDN?'))

if (len(sys.argv)>1):
    replyFromInstrument = sendCommand(sys.argv[1])
    sys.stdout.write(replyFromInstrument)
else:
    test()
```

**Şekil 2.17** Python üzerinden VISA kullanarak cihaz ile haberleşme

Şekil 2.17'de VISA sürücüleri işletim sistemi üzerinde ve PyVisa eklentisi Python üzerinde kurulduktan sonra, cihazla haberleşme ile ilgili temel bir kod örneği görülmektedir.

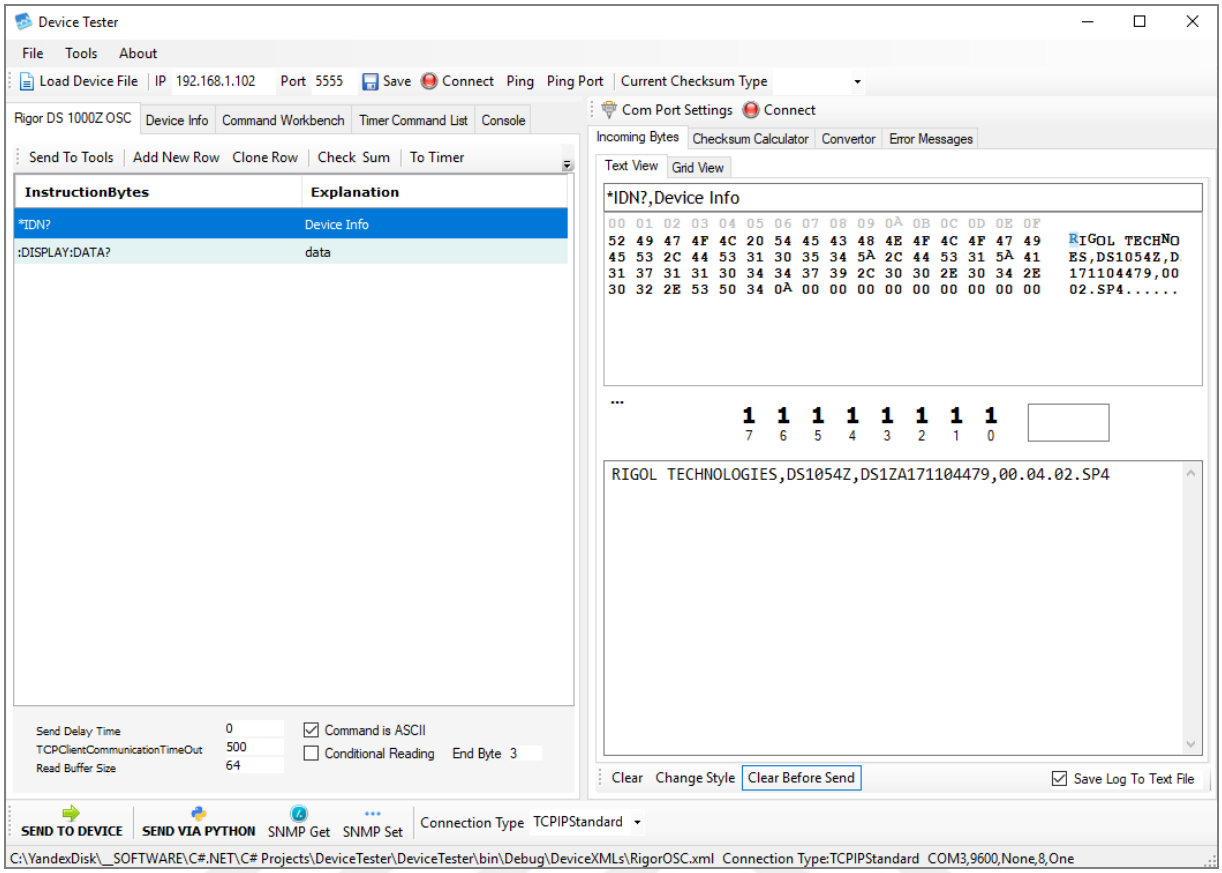
Şekil 2.18'de ise ölçüm cihazları ile haberleşmenin programlama açısından

katmanları gösterilmiştir. Buna göre VISA sürücüsü olmaksızın da cihazla haberleşmek mümkündür. Bunun için SCPI standardına göre yazılan birer ASCII metinden ibaret olan SCPI komutları telnet veya .NET'in TCPClient sınıfı kullanılarak cihazın IP adresi ve standart olarak kullanılan 5024 veya 5025 portu üzerinden cihaza gönderilir




**Şekil 2.18** SCPI, VISA, IVI Driver ilişkisi

Şekil 2.18'de "*IDN?" komutu standart bir SCPI komutu olup cevap olarak cihazın marka, model, seri numarası gibi bilgilerini gönderir. Low Level I/O seviyesinde komut doğrudan TCP/IP haberleşme yapan soket üzerinden gönderilmiştir. VISA seviyesinde işletim sistemi üzerinde VISA kuruludur ve komut VISA sürücüsü üzerinden gönderilmiştir. IVI Drivers seviyesinde, alt katmanları bilmeye gerek olmadan cihazı üreten firmanın VISA sürücüsü kullanarak yazmış olduğu "GetDeviceInfo()" metodu kullanılmıştır. VISA seviyesinde soket nesnesi ile, IVI Driver seviyesinde ise bir katman daha yukarıda çalışarak, "*IDN?" ile de yüzleşilmemektedir.



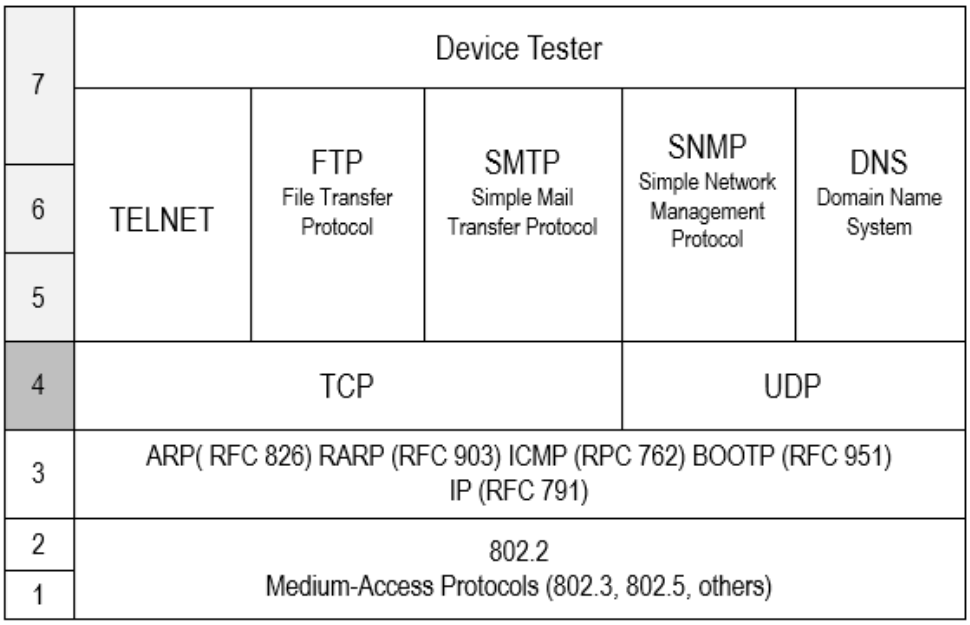

**Şekil 2.19** Program üzerinden SCPI komutları gönderilerek cihazla haberleşme

Şekil 2.19'da SCPI komutunu en alt seviyede cihaza göndermenin bu tezde geliştirilen programdaki örneği görülmektedir.

Porta dikkat edilirse, 5025 standart port numarası yerinde 5555 yazılı olduğu görülür. Bu Rigol firmasının tercihinden kaynaklı bir durumdur.

Şekil 2.19'daki örnekte işletim sistemi üzerinde VISA sürücüsü kurulumu mevcut değildir ve haberleşme Şekil 2.18'de gösterildiği gibi TCP soket üzerinden gerçekleşmekte ve SCPI komutları kullanılmaktadır.

## 2.7. Programın OSI Yapısında Yeri



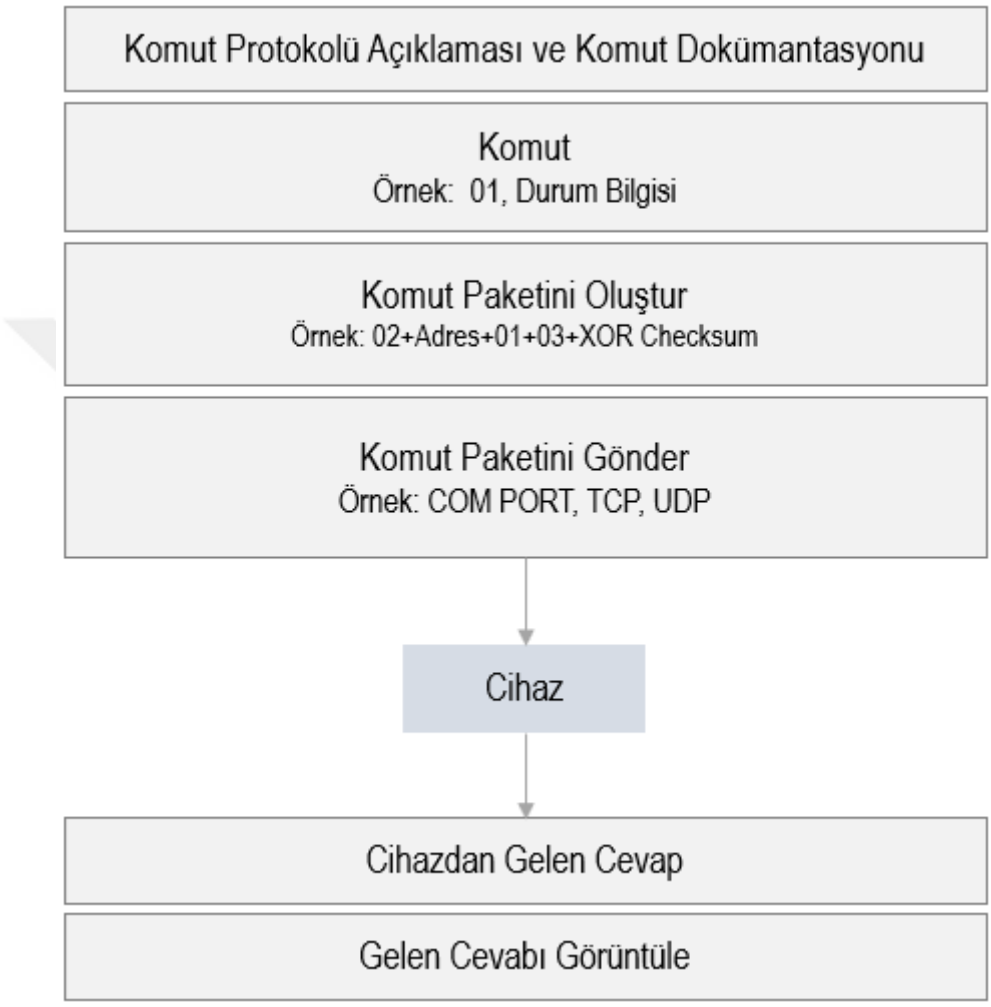

**Şekil 2.20** OSI modeli yapısında uygulamanın konumu

Şekil 2.20'de Open Systems Interconnection (OSI) modelinin yapısında, TCP, UDP ve SNMP'nin konumları görülmektedir. Bu tezde geliştirilen program 4.katmandan başlayarak yukarısını kapsar.

C# dilinin TCPClient ve UDPClient sınıfları, Python'un socket sınıfı 4.katmandadır.

## 2.8. Cihazla Haberleşmede Genel Çalışma Yapısı

Şekil 2.21 bir cihazla bir protokol üzerinden haberleşmenin yapısını göstermektedir. Başlama noktası cihazın kullandığı protokol bilgisi ve sonra cihaz komutları ile ilgili dökümandır.




**Şekil 2.21** Cihazla haberleşme çalışmasının blok yapısı

Eğer kullanılan protokolde checksum varsa, ilgili checksum hesabını yapacak bir araç gereklidir.

Örnek olarak 0x01 şeklinde bir komutun cihazın durum bilgisini gönderdiğinin bilinmesi gerekir. Sonra protokole göre komut paketinin belirtilen noktalarına başlama, bitiş, adres, komut byte değerlerinin konulması ve sonra tüm paketin checksum değerinin pakete eklenerek, seri haberleşme portu, TCP/IP gibi bir arayüz üzerinden cihaza gönderilmesi gereklidir.

Cihazdan cevap alındıktan sonra, bu cevabın gösterilmesi ve cevabın her bir byte değerinin analiz edilebilmesi gereklidir.

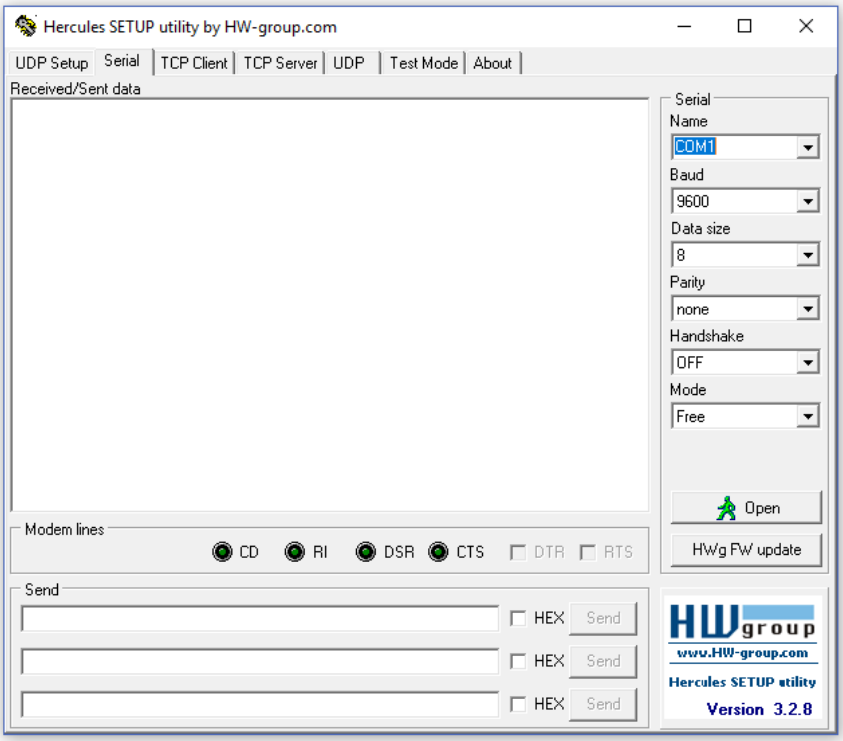
Buna göre bir cihazla haberleşme için minimum gereklilikleri listelersek;

- Kullanılan protokol ve komutları açıklayan cihaz dökümanı
- Kullanılan bir checksum varsa, bu checksum hesabını yapan bir hesaplayıcı
- Komut paketini cihaza gönderecek bir program

# 3. BENZER PROGRAMLARIN İNCELENMESİ

## 3.1 HW-Group Hercules Multiformat Terminal Yazılımı

Hercules HW-Group [21] adlı IP arayüzlü sensörler geliştiren ve üreten bir firmanın geliştirdiği bir programdır.




**Şekil 3.1** Hercules programının arayüzü

Hercules programının Şekil 3.1'de görülen arayüzünden de anlaşılacağı üzere, seri port haberleşme, UDP haberleşme, TCP Client ve Server [16,17] olarak kullanılabilme gibi özellikleri vardır.

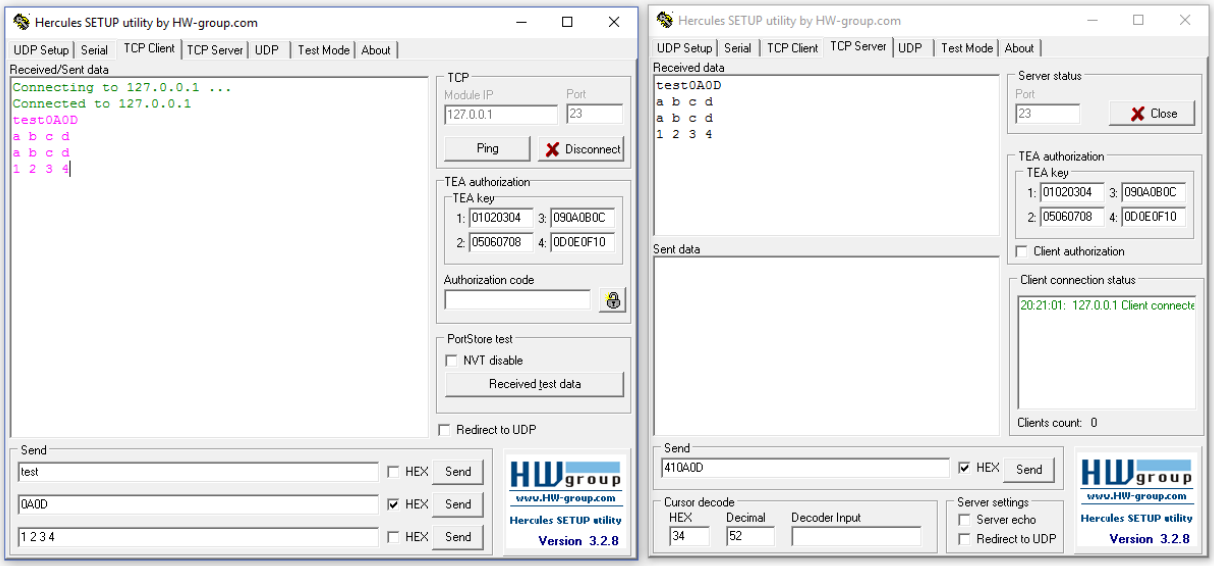
Bağlantı sağlandıktan sonra metin kutusuna yazılan mesajlar karşıya gönderiliyor. Mesajlar heksadesimal olarak da yazılabiliyor.

Hercules programının sunucu olarak çalışma özelliği de olduğundan, programların incelenmesinde haberleşme için sunucu olarak Hercules programı kullanılacaktır.

Hercules programında her bir özellik ayrı bir sekmededir. Örneğin seri haberleşme yapılacaksa "Serial Port" sekmesi seçilir.

Her bir sekmede ilgili bağlantı parametreleri mevcuttur. IP, Port gibi parametreler girilip bağlantı başlatılır.

Komut gönderimi birer satırdan ibaret üç adet metin kutusu üzerinden yapılır. Komutları HEX formatta gönderme seçeneği de mevcuttur. Yani hem ASCII hem HEX formatta komut gönderilebilir.




**Şekil 3.2** Hercules programı üzerinden haberleşme

**Hercules Programının Sunucu Olarak Kullanılması:** TCP Server sekmesinde "Listen" butonuna tıklayınca Hercules programı belirtilen port üzerinden sunucu olarak dinlemeye başlar. Yukarıdaki örnekte port 23 olarak görülüyor.

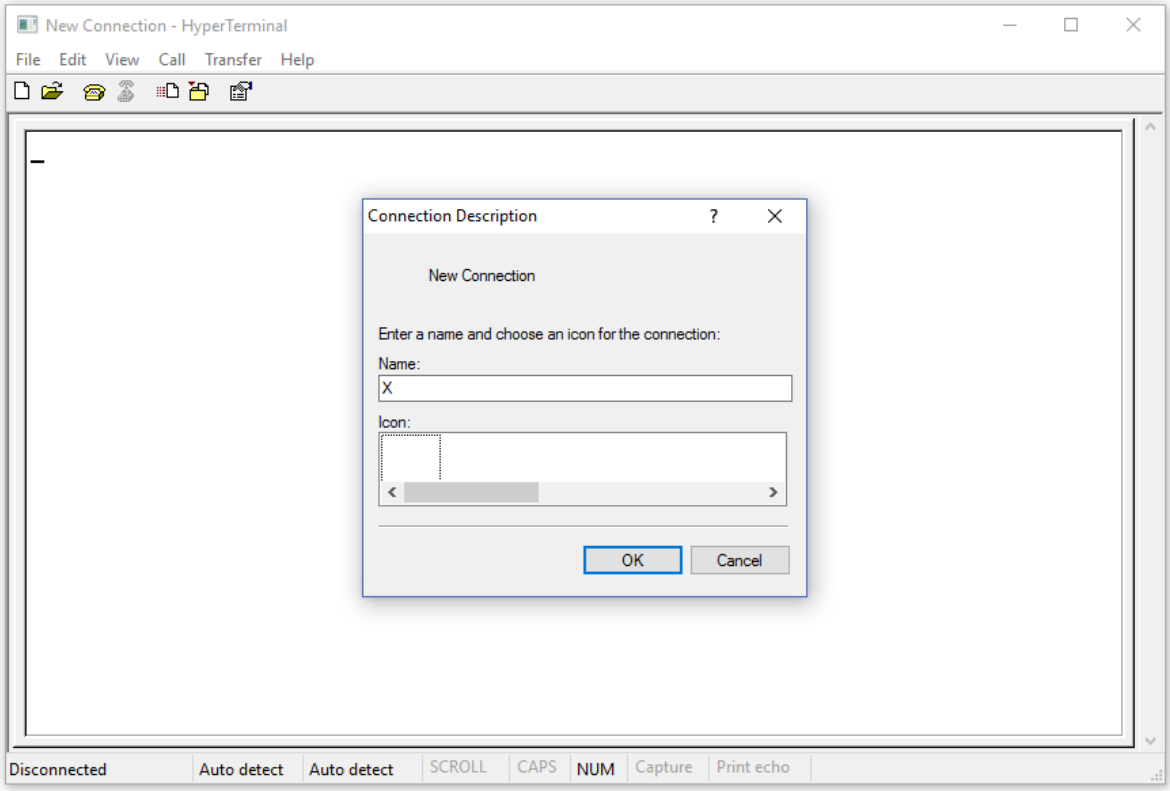
**Hercule Programının İstemci Olarak Kullanılması:** Şekil 3.2'deki haberleşme örneğinde Hercules programı çalıştırıldıktan sonra, ikinci bir Hercules programı açılıp, istemci olarak kullanılmıştır. Bunun için program açıldıktan sonra TCP Client sekmesi seçilir, IP 127.0.0.1 ve Port 23 olarak girilerek "Connect" butonun basılır ve birinci Hercules programına bağlanılmış olur. Bu aşamadan sonra istemci program sunucuya veri göndermeye hazırdır.

Şekil 3.2'deki örnekte, ilk olarak test mesajı gönderiliyor. İkinci satırda 0A0D HEX mesajı gönderilmiştir. Bu mesaj aslında yeni satır (newline) karakteridir. Önce "test", ardından HEX onay kutusu seçili haldeyken "0A0D" ve ardından iki kez "a b c d" ve sonra "1 2 3 4" gönderildiğinde oluşan durum Şekil 3.2'de görülüyor.

Hercules programının komut gönderme işleyişi bu şekilde olup komutları ASCII ve HEX gönderme özelliği mevcuttur. Ancak komutlar için yalnızca üç adet kutu ayrılmıştır ve komutları farklı uygulamalar için saklama özelliği yoktur. Gelen veri arayüzde ASCII olarak, oldukça basit bir şekilde gösterilmekte olup gelen byte dizisinde örneğin üçüncü byte değeri nedir, bu byte değerinin ikinci bit değeri nedir gibi sorulara bu arayüzde cevap verilmez.

### 3.2 HyperTerminal Terminal Emülasyon ve İletişim Programı

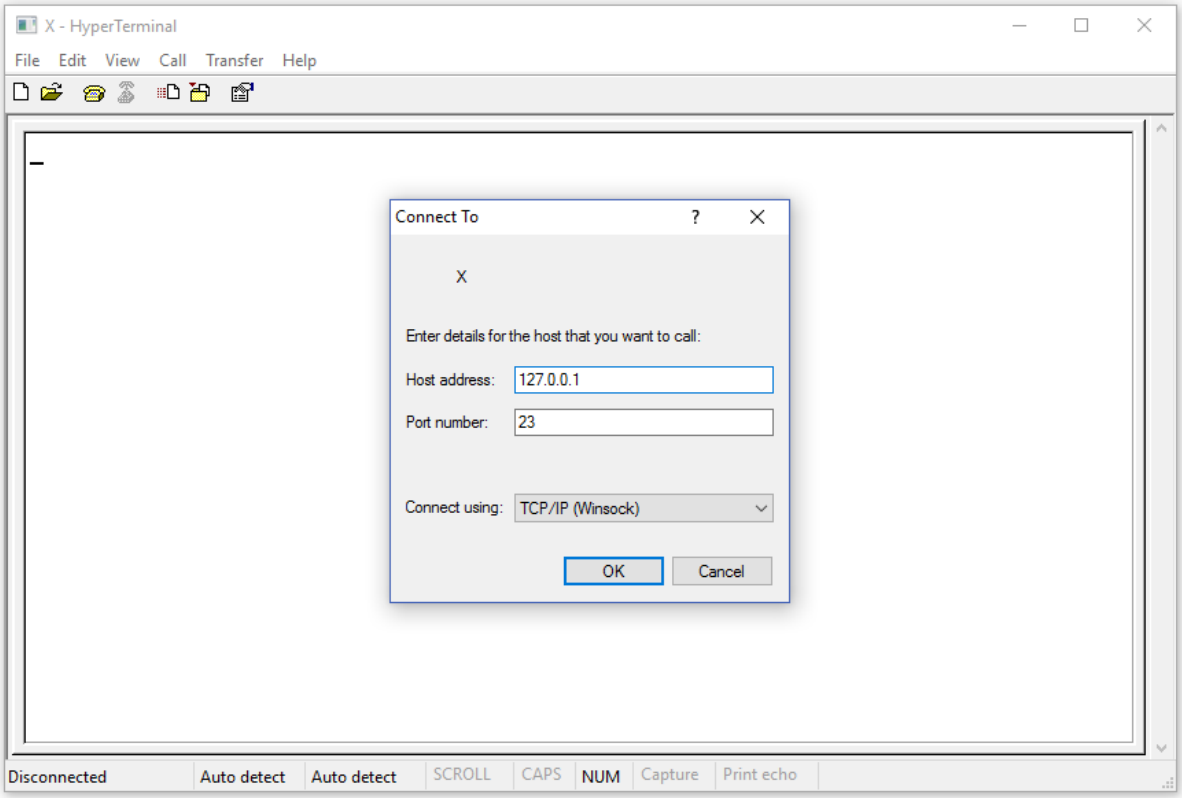
HyperTerminal [20] ilk olarak Windows 95 içinde geldi ve kullanılmaya başlandı. Ancak Windows Vista ile birlikte kaldırıldı. Buna rağmen, Windows XP'nin Windows/System32 klasöründe mevcut iki adet dosyadan (hypertrm.dll, hypertrm.exe) ibaret bir uygulama olduğundan, buradan kopyalanarak Windows 10 üzerinde de kullanılabiliyor. HyperTerminal seri port üzerinden bir haberleşme gerekliliğinde en yaygın kullanılan programdır [2].




**Şekil 3.3** HyperTerminal programı ile yeni bir bağlantı oluşturma

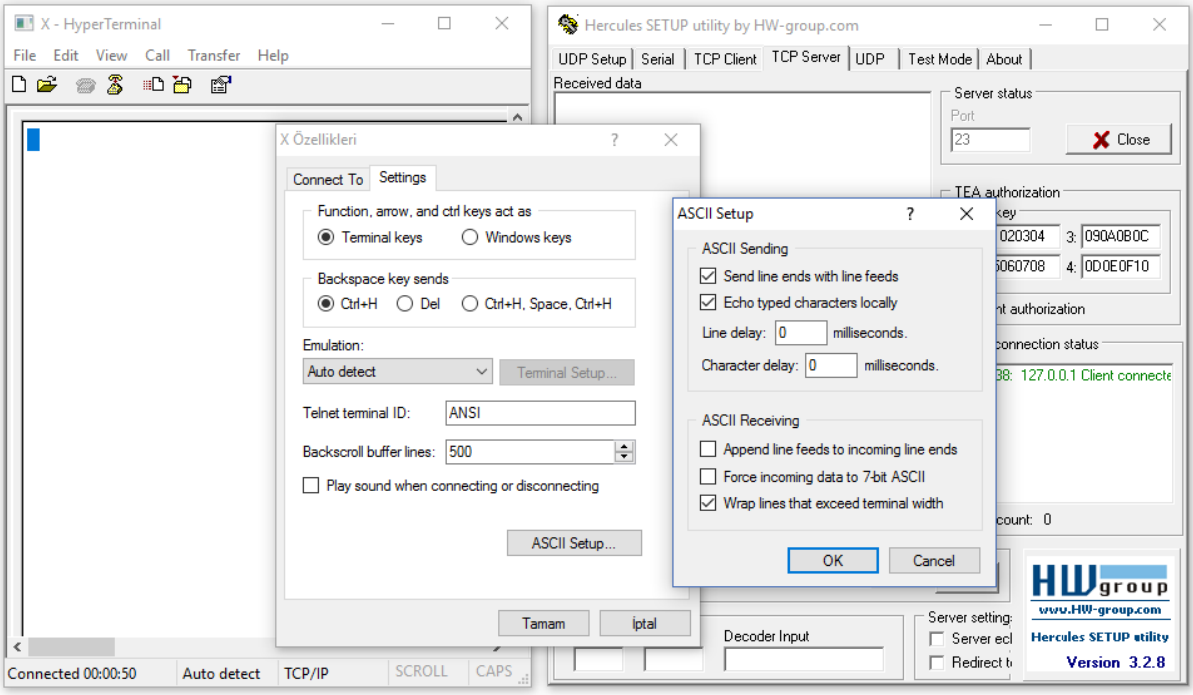
HyperTerminal başlatıldığında ilk olarak bağlantı tanımlama penceresi açılır. Şekil 3.3'te görüldüğü gibi bu bağlantıya bir isim verip OK butonuna bastıktan sonra Şekil 3.4'te görülen şekilde bağlantının tipi ve detayları girilir.

Bu bağlantı bilgileri bir dosyaya kaydedilir ve yeniden bağlantı kurulması gerektiğinde aynı şeylerin tekrar girilmesine gerek kalmaz. Yapılması gereken sadece o dosyanın açılmasıdır.



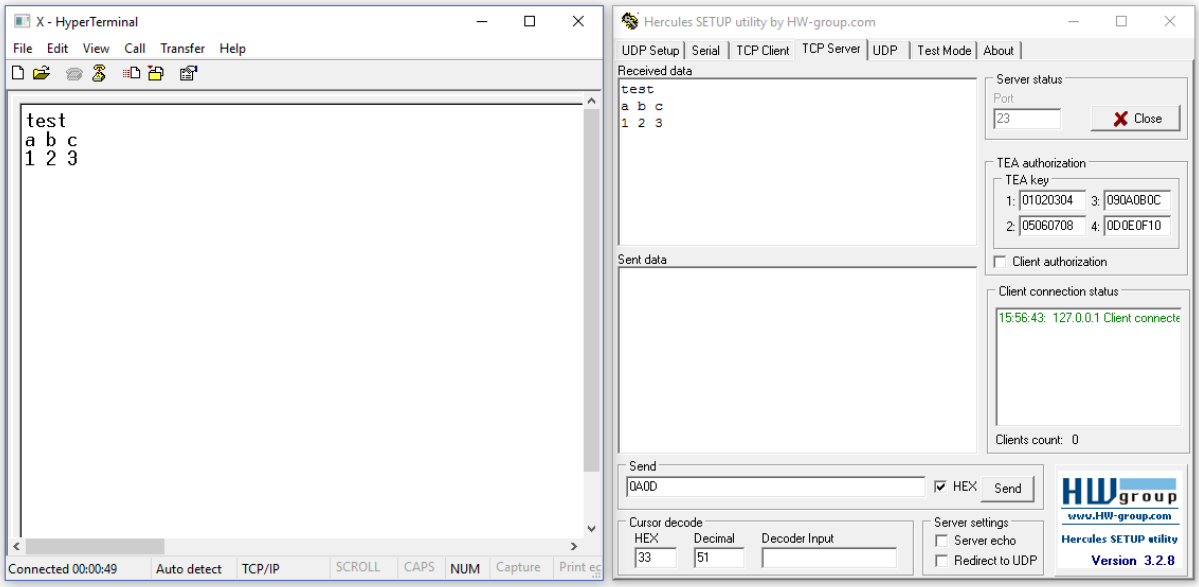

**Şekil 3.4** HyperTerminal programı TCP/IP parametreleri girişi

Bu çalışmada geliştirilen uygulamada haberleşme bilgileri komut dosyasının içinde olduğundan, yeni bir bağlantı oluşturma ve bu bağlantıyı kaydetme işlemine gerek yoktur. Eğer HyperTerminal programının, bu bağlantı dosyası içerisinde komutları da saklama özelliği olsaydı, HyperTerminal en çok ihtiyaç duyulan özelliklerden birini sağlamış olurdu.




**Şekil 3.5** HyperTerminal programı konsol ayarları

İletişim esnasında kullanıcının karşıya gönderdiği komutları görebilmesi için HyperTerminal üzerinde ayar yapılması gerekmektedir. Şekil 3.5'te bu ayarların yapıldığı pencereler görülüyor. Şekil 3.6'da HyperTerminal üzerinden Hercules sunucuya veri gönderilmesi örneği görülmektedir.

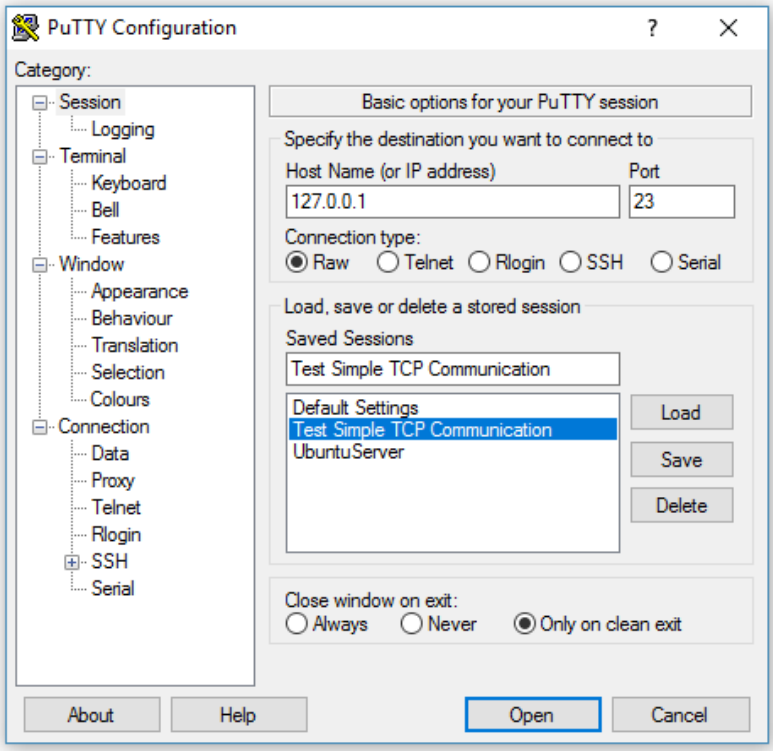



**Şekil 3.6** HyperTerminal programı ile haberleşme örneği

Şekil 3.6'da görüldüğü gibi, HyperTerminal komut gönderme arayüzü ASCII bir metin editörü şeklinde olup, bu arayüzden Hercules programındaki gibi HEX formatta veri gönderme özelliği bile yoktur. Komutları saklama özelliği, gönderilmiş komutlar ve gelen cevaplardan oluşan metnin dosya olarak kaydedilmesinden ibarettir.

### 3.3 Putty Seri Konsol ve Terminal Emülatörü

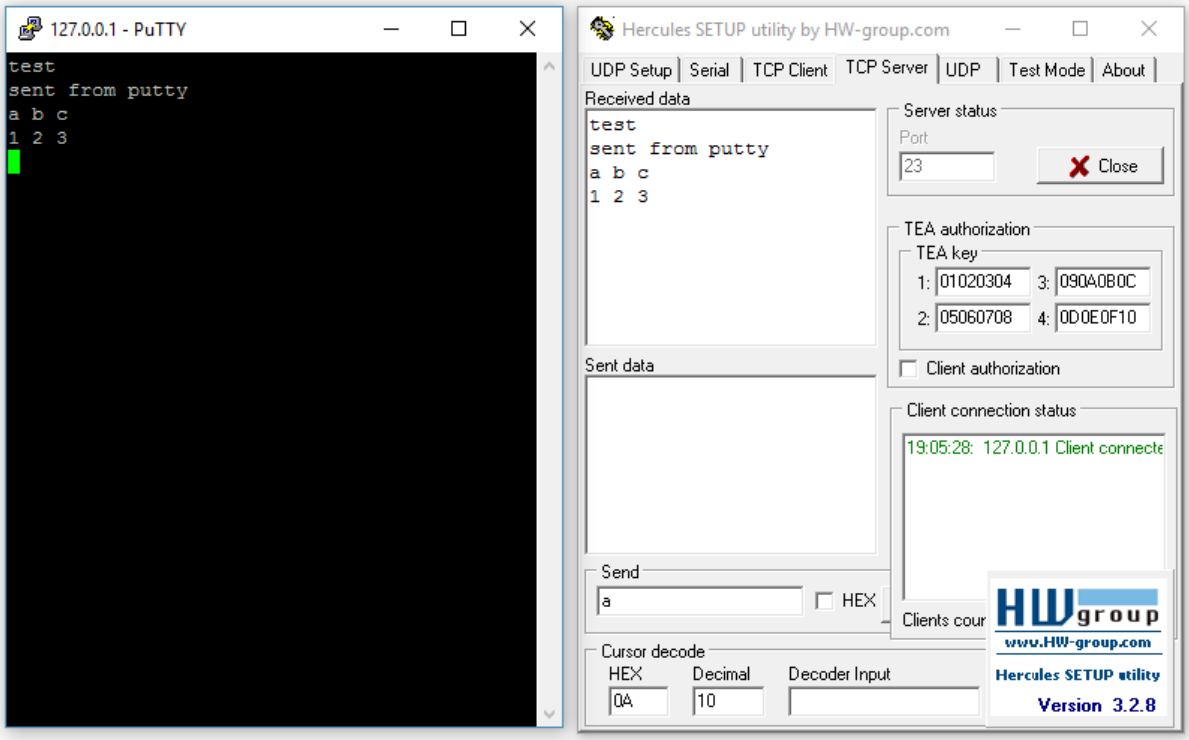
1999'da Simon Tatham tarafından yazılan, SCP, SSH, Telnet, Serial Port ve raw socket desteği olan bir programdır [22]. Uzaktan Linux işletim sistemine terminal olarak bağlanmak gerektiğinde en çok kullanılan programdır.




**Şekil 3.7** Putty programı iletişim ayarları

Şekil 3.7'de Putty programının ana ekranında çok farklı iletişim protokollerini desteklediği görülmektedir. Bu yönüyle bu tezde geliştirilen uygulamadaki amacın bir kısmını gerçekleştiriyor. Ancak gönderilen verinin loglanması, komutların kaydedilmesi, checksum hesaplanması, gelen cevabın ASCII, HEX, desimal formatlarda görünmesi, bir byte bilgisinin bit görünümü gibi özellikler bu programda bulunmamaktadır. Cihaz üzerinde test yapmak için Putty programını kullananlar, bu

gereksinimleri farklı programlar kullanarak gidermelidirler.



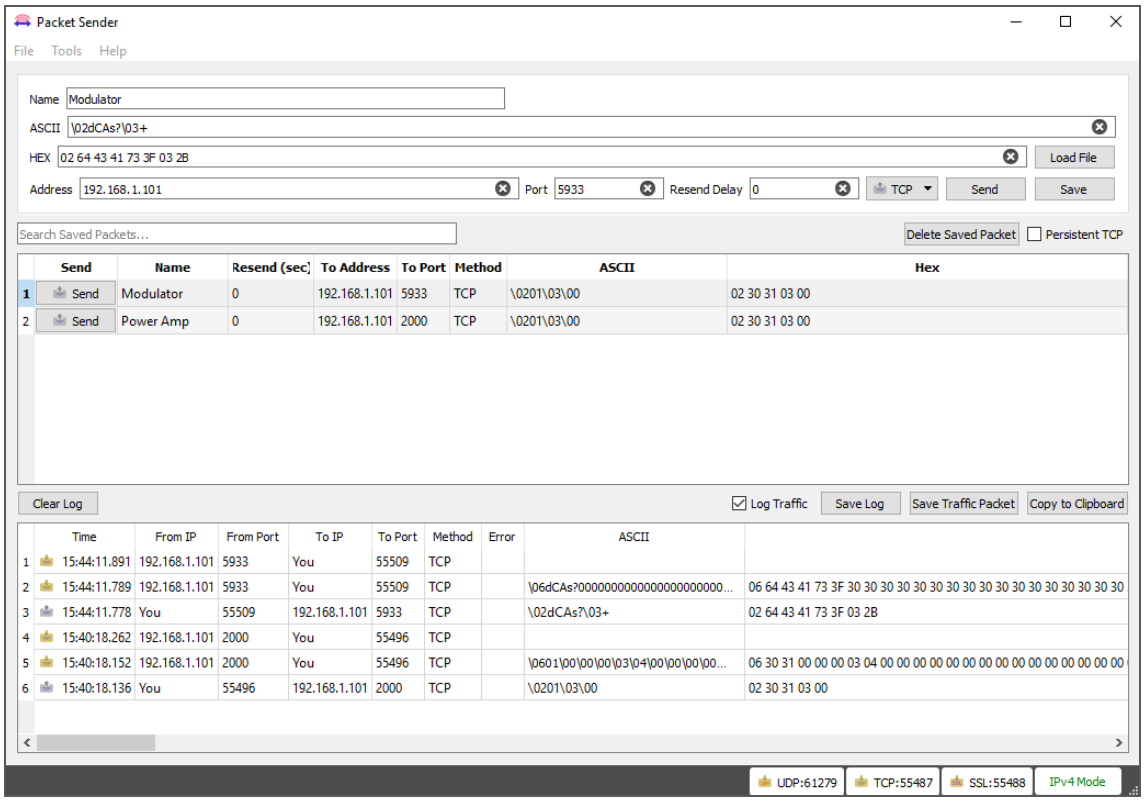

**Şekil 3.8** Putty programı ile haberleşme

Şekil 3.7'de bağlantı seçildikten sonra "Open" butonuna basıldığında Şekil 3.8'de sol tarafta görülen pencere açılır. Veri bu pencere üzerinden sağ taraftaki sunucu programına gönderilmektedir.

Putty'nin veriyi gönderme ve alma temel fonksiyonlarını yerine getiren basit bir arayüzü olduğu görülmektedir. Gelen veriyi HEX formatta görme özelliği eklenmemiştir. Komutları saklama özelliği ise ekranda mevcut görülen metni hafızaya kopyalamaktan ibarettir.

### 3.4 PacketSender Network Test Programı

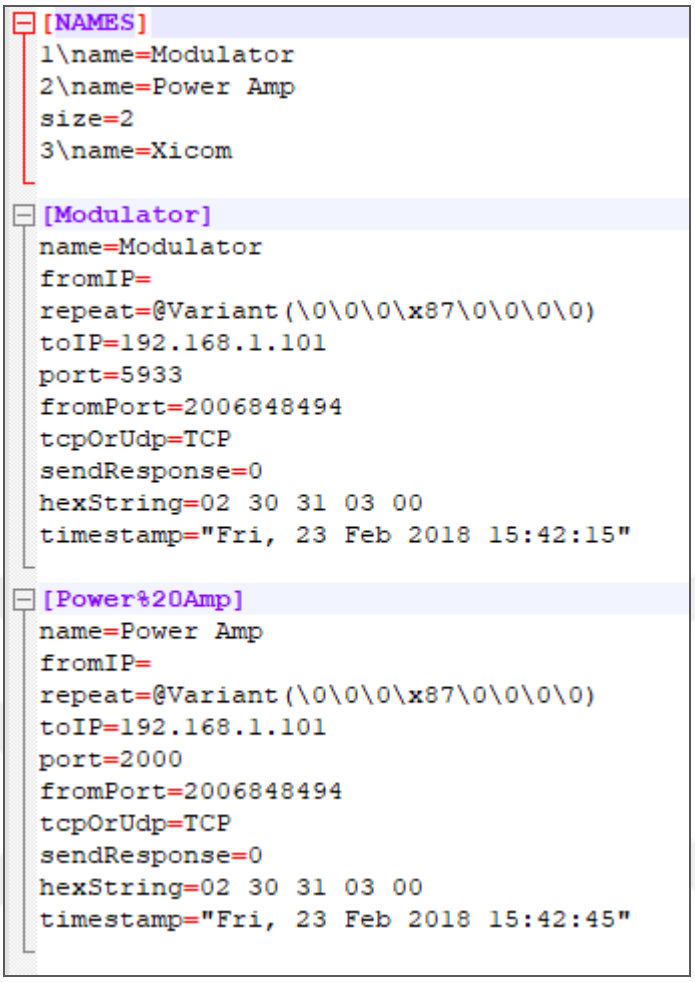
Dan Nagle tarafından ağ analizinde kullanılmak üzere geliştirilmiş bir programdır [23]. TCP, UDP ve SSL protokollerine destek vermektedir.




**Şekil 3.9** Packet Sender programının arayüzü

Program komutları HEX ve ASCII olarak görüntüler. Komutlar TCP, UDP, SSL bağlantı şekillerinden seçili olana göre gönderilir. Gelen cevaplar programın alt kısmında satırlar halinde ASCII ve HEX olarak listelenir. Ağ analizi amaçlandığından, programda gelen, giden port parametreleri de dikkate alınmıştır.

Farklı IP adreslerine sahip birimler programa eklenebilir. Bu şekilde oluşan liste kaydedilebilir. Program bu listeyi bir *.ini dosyasına kaydeder. Bu dosyanın yapısı Şekil 3.10'da gösterilmiştir.




**Şekil 3.10** Packet Sender programının komut ve ayarlar için dosya yapısı

Packet Sender programının bu çalışmada geliştirilen uygulamaya benzer özellikleri olduğu görülmektedir. Ancak Şekil 3.10'da görüldüğü gibi, her bir IP için tek bir komut saklayan, komutu standart olarak bir programın temel ayarlarının saklanması için kullanılan tek bir ini dosyasında saklayan, bir cihazla çalışılması gerektiğinde, gereksiz yere diğer bütün cihazları da ana ekrandaki bir listede sürekli gösteren, komutlarla IP ve port parametrelerini bir arada gösteren bir yapısı vardır.

### 3.5 Mevcut Programların Eksiklikleri

Cihazla haberleşmede en çok kullanılan programların incelenmesi sonucu, bu tez çalışmasının giriş kısmında belirtilen bazı özelliklerin hiçbirinde olmadığı görülmüştür.

### 3.5.1. Komutların Kaydedilmesi Ve Düzenlenmesi

Cihazların komutlarını kaydetme ve düzenleme ile ilgili bir özellik bu programlarda mevcut değildir. Öte yandan cihaz üzerinde çalışan geliştiriciler, sürekli farklı komutları denemek ve komutların doğru çalıştığından emin olmak durumundadır. Bu komutlar tekrar gerekli olduğunda onlara kolay ulaşmak ve kolaylıkla kullanmak gereklidir. Bu programlarla çalıştıklarında, geliştiriciler komutları ayrı metin dosyalarında saklamak ve bir komutu kullanmak gerektiğinde, o dosyayı bulup, açıp, oradan komutu kopyalayıp yapıştırarak haberleşme ve test sürecine devam etmek durumundadırlar. Bu koşullar altında, komutların nasıl saklanacağı her kullanıcının kendisinin çözmesi gereken bir sorundur. Bütün bunlar ciddi zaman kaybına neden olmaktadır.

### 3.5.2. Komut Paketinin Hazırlanması ve Checksum Hesabı

Cihaz üzerinde çalışanlar komut protokolüne bağlı olarak, kimi zaman bir byte dizisinin checksum sonucunu hesaplayıp bu sonucu da protokolde belirtilen şekilde komut paketine dahil ederek cihaza göndermek durumundadır. Bu programların hiçbirinde checksum hesaplama özelliği yoktur. Geliştiriciler bunu, farklı kaynaklar kullanarak kendileri çözmek zorundadır.

### 3.5.3. Gelen Cevabın Analizi

Cihaza komut başarılı bir şekilde gönderilip, cevap geldiğinde, geliştiricilerin önünde cevabı analiz etme işi vardır.

Bu noktada, gelen cevabın hem HEX hem ASCII formatta aynı anda görülmesi, gelen cevabın byte dizisi şeklinde görülmesi, herhangi bir byte değerine bakıldığında onun kaçıncı byte olduğunun bilinmesi, o byte değerinin bitlerinin anında görülmesi gibi ihtiyaçlar söz konusudur.

Yukarıdaki programlar bu tip ihtiyaçları karşılamada yetersizdir. Bu programlarda gelen cevap ayrı bir program kullanılarak incelenmelidir.

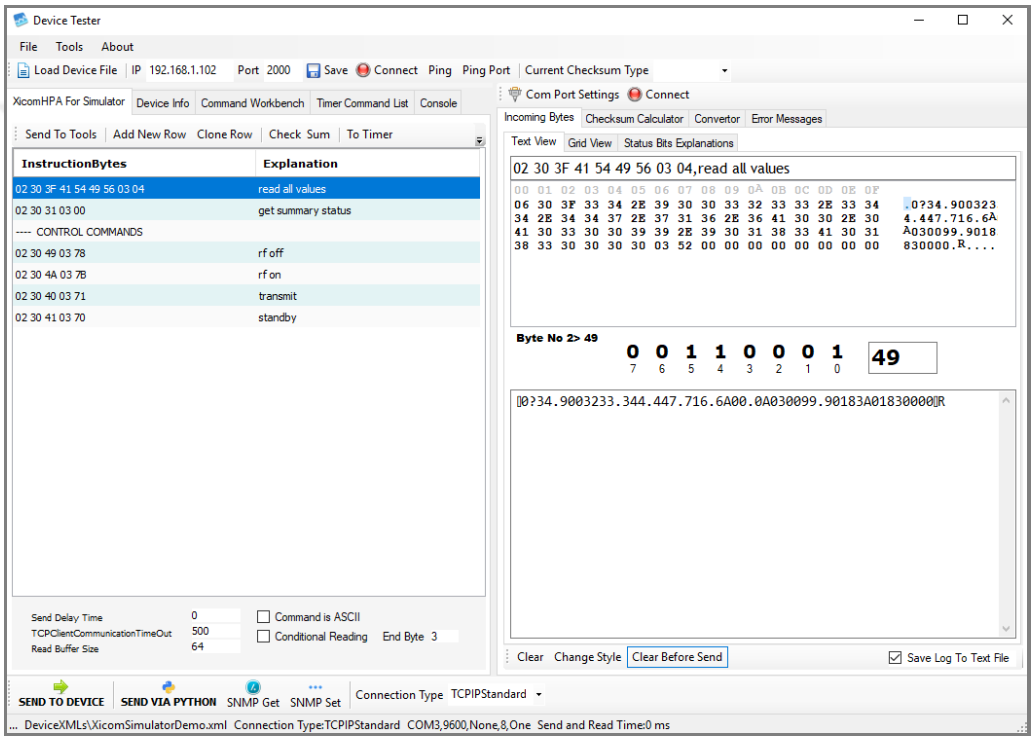
### 3.5.4. Sıralı Otomatik Komut Gönderme

Bu programlarda, komutlar kayıt edilmediğinden, onları belirli zaman aralıklarında cihaza göndermek gibi bir kavram da söz konusu değildir.

# 4. YENİ BİR HABERLEŞME YAZILIMININ GELİŞTİRİLMESİ

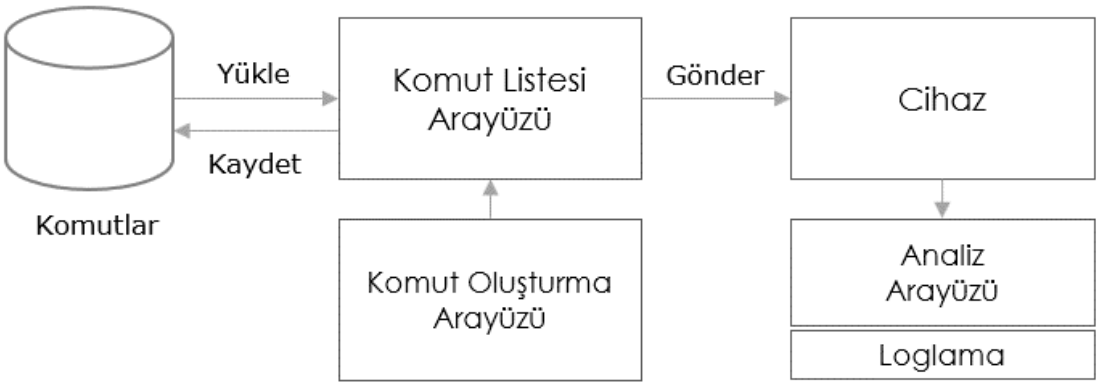
## 4.1. Programın Genel Yapısı

Şekil 4.1'de geliştirilen yazılımın ilk açılıştaki arayüzü görülmektedir. Arayüzde sol taraf komut dosyalarını yükleme, en sık kullanılan bağlantı tipi olan TCP/IP için IP ve Port ayarları ve bu ayarların doğruluğunu test, komutlar üzerinde değişiklikler yapma, zamanlayıcı için komut listesi ve komutların cihaza gönderilmesi ile ilgili özellikleri barındırır.



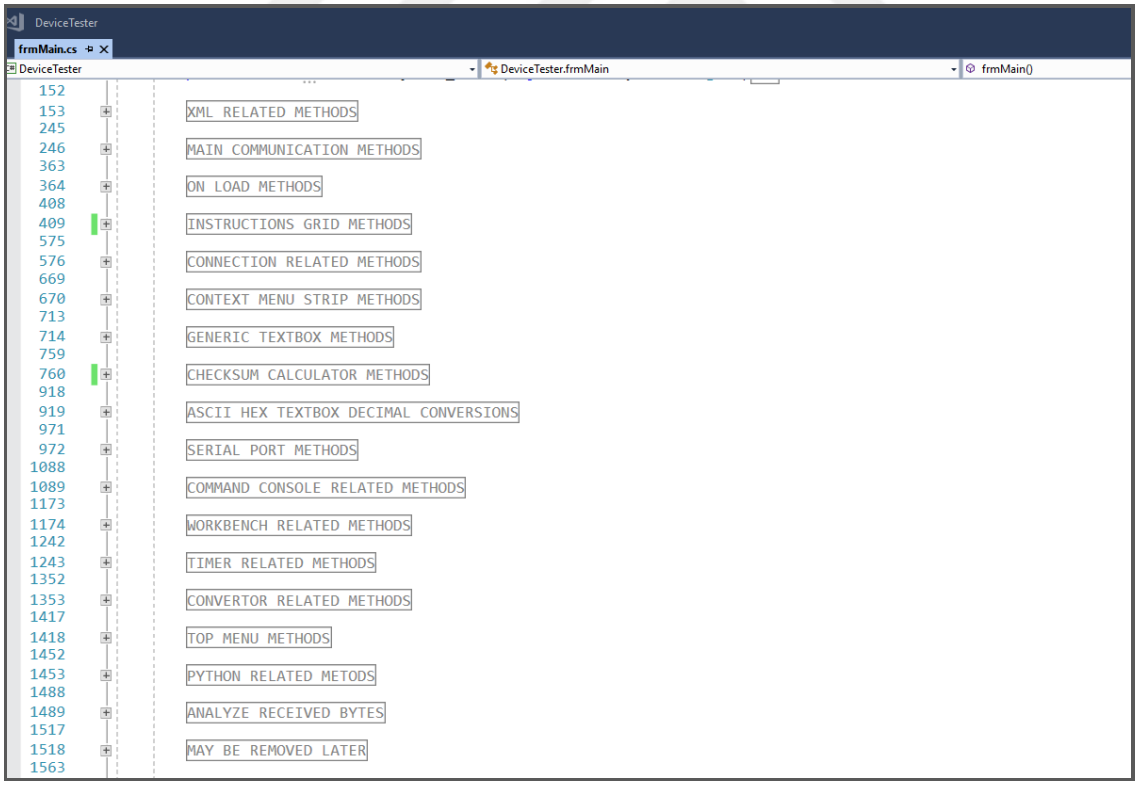

**Şekil 4.1** DeviceTester programının arayüzü

Şekil 4.1'de sağ taraf ise gelen cevabın gösterimiyle ilgili özellikleri ve checksum hesaplayıcıyı barındırır.



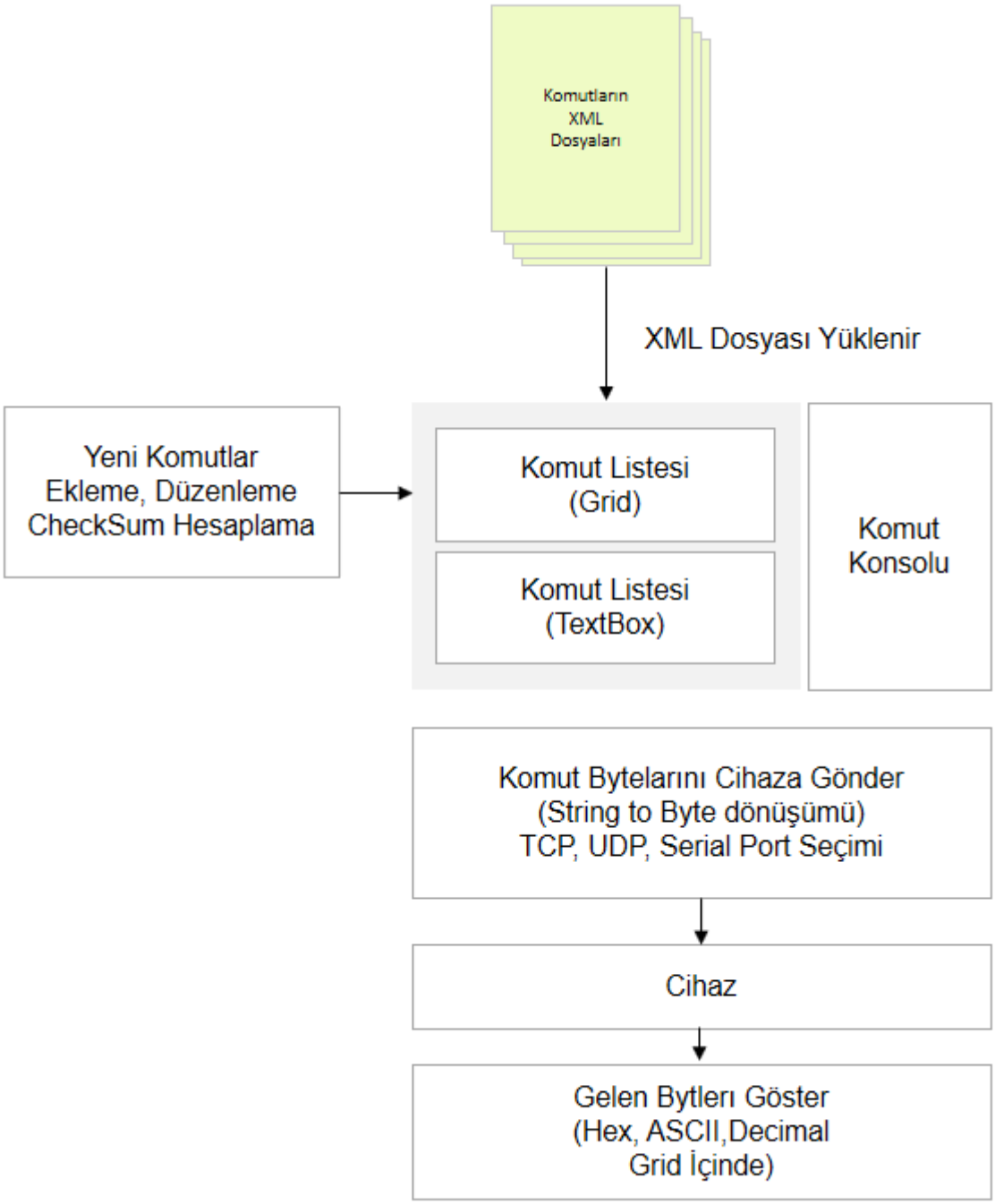

**Şekil 4.2** DeviceTester programının şematik genel yapısı

Şekil 4.2'de programın genel işleyişi gösterilmiştir. Komutlar dosyalar şeklinde tutulur ve ihtiyaç duyulduğunda komut listesi arayüzüne yüklenir. Yüklemeden sonra, komutlar üzerinde yeni komutlar oluşturma ve test etme şeklinde çalışma yapıldıkça bunlar kaydedilir. Cihaza gönderilen komutlardan gelen cevaplar arayüzde gösterilir ve loglanır.




**Şekil 4.3** DeviceTester programının ana formunun kod görünümü

Şekil 4.3'te programın ana kod dosyası olan frmMain.cs'teki ana kod blokları gösterilmiştir.

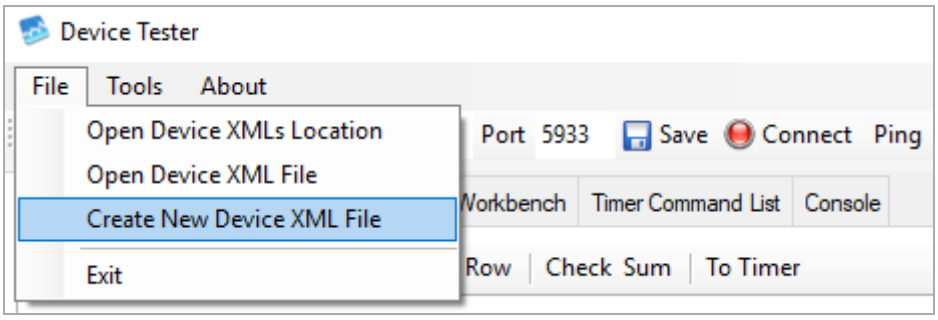



**Şekil 4.4** DeviceTester programının detaylı işleyiş şeması

Şekil 4.4'te komut oluşturma arayüzünün yeni komut ekleme, komutları düzenleme ve checksum hesaplama özellikleri içerdiği görülüyor. Şekil 4.2'de gösterilen "Komut Listesi Arayüzü"nün ise, grid ve metin kutusu şeklinde iki farklı yapıda olduğu görülüyor.

### 4.2. Komut Dosyalarının Yönetimi

Komutlar XML formatındaki dosyalarda saklanmaktadır. XML dosyaları ile ilgili işlemler Şekil 4.5'te görülen "File" menüsü kullanılarak yapılmaktadır.




**Şekil 4.5** Yeni bir cihaz dosyasının oluşturulması

Bir cihazla ilgili çalışmaya başlamak için ilk adım o cihaz için bir XML dosyası oluşturmaktır. Şekil 4.5'te "File" menüsü üzerinde "Create New Device XML File" kullanılarak yeni bir dosya oluşturulur.

Program kendisi ile, yani "DeviceTester.exe" ile aynı klasörde bulunan "DeviceXMLs" klasörü içerisinde girilen isimde bir XML dosyası oluşturur. Dosya üzerinde bu noktadan sonra çalışmaya başlanır.

"File" menüsü üzerinde "Open Device XMLs Location" tıklanarak, bütün cihaz XML dosyalarının bulunduğu klasöre doğrudan ulaşılabilir.

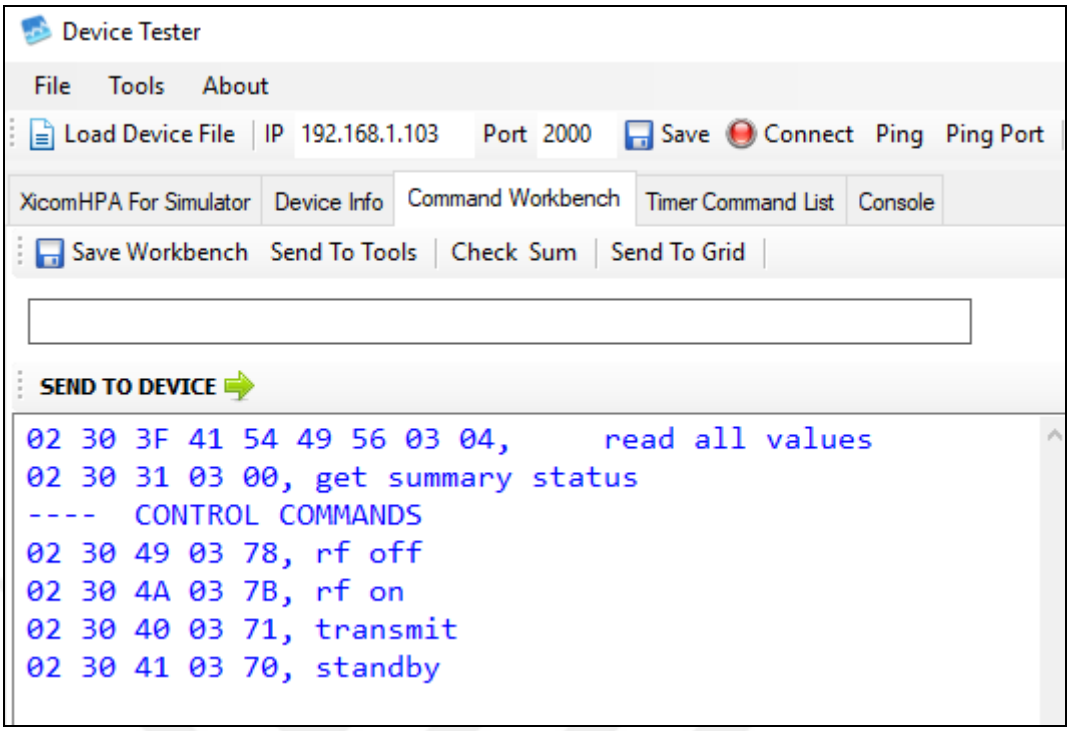
"File" menüsü üzerinde "Open Device XML File" ise mevcut cihaz XML dosyalarından birini seçerek programa yüklemek için kullanılır.

```xml
<?xml version="1.0" encoding="utf-8"?>
<DeviceInstructions>
    <DeviceInfo>
        <DeviceName>Xicom High Power Amplifier</DeviceName>
        <Explanation>Uplink HPA</Explanation>
        <IP>192.168.1.104</IP>
        <Port>2000</Port>
        <CheckSumType>CheckSum,1 Byte</CheckSumType>
        <InstructionStructure>
            02 + Address+ Command Bytes + 03 + XOR Checksum
        </InstructionStructure>
    </DeviceInfo>
    <WorkbenchData>
        <ID>1</ID>
        <Instructions>
            02 30 3F 41 54 49 56 03 04, read all values
            02 30 31 03 00, get summary status
            02 30 49 03 78, rf off
            02 30 4A 03 7B, rf on
            02 30 40 03 71, transmit
            02 30 41 03 70, standby
        </Instructions>
    </WorkbenchData>
    <Instruction>
        <ID>1</ID>
        <Order>2</Order>
        <InstructionBytes>02 30 3F 41 54 49 56 03 04</InstructionBytes>
        <Explanation>read all values</Explanation>
        <CheckSum>1</CheckSum>
    </Instruction>
    <Instruction>
        <ID>2</ID>
        <Explanation>get summary status</Explanation>
        <InstructionBytes>02 30 31 03 00</InstructionBytes>
        <Order>3</Order>
    </Instruction>
    <Instruction>
</DeviceInstructions>
```

**Şekil 4.6** Komut dosyasının XML yapısı

Şekil 4.6'da görüldüğü gibi XML dosyasının yapısında, üç ana kısım bulunmaktadır.

Birinci kısım DeviceInfo'dur. Burada cihazla ilgili cihaz tanımı, IP, Port, varsa checksum tipi ve komut yapısı parametreleri bulunur. IP ve port program tarafından aktif olarak kullanılmakta olup, diğer parametreler bilgi amaçlı saklanmaktadır.




**Şekil 4.7** Komutların metin şeklinde saklandığı kısmın görünümü

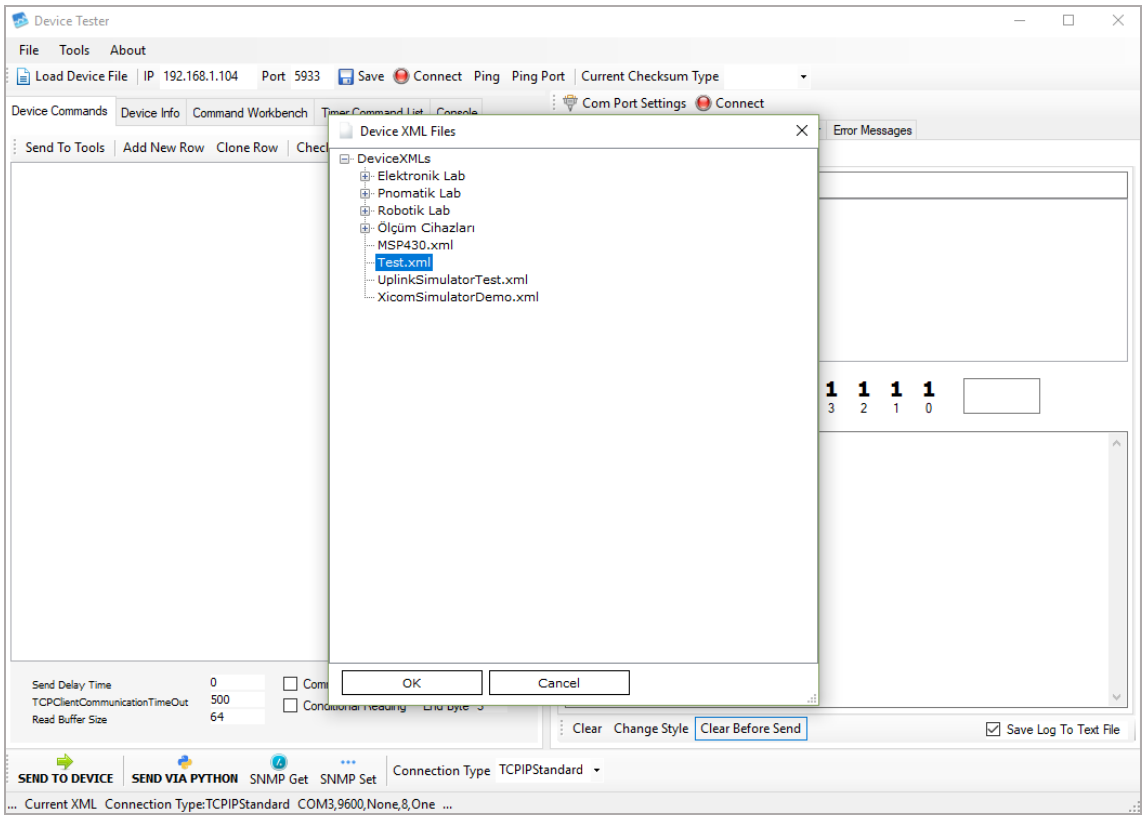
İkinci kısım WorkbenchData olup, burada Şekil 4.7'de görülen "Command Workbench" sekmesindeki bilgiler saklanır. Bu kısım, komutlar üzerinde bir metin editörü içinde daha esnek bir şekilde çalışmayı mümkün kılar.

Üçüncü kısım "Instruction" parametrelerinin bulunduğu kısımdır. Her komutun detayları bir "Instruction" XML etiketi içerisinde tutulur. Bu etiket içerisindeki parametrelerin açıklaması ise sırası ile şu şekildedir:

- ID programın her komut üzerinde kaydetmek, güncellemek gibi işlemleri yapabilmesi için gereklidir.
- Order listedeki sıralama içindir. Komutların listeleme sırası için order parametresi kullanılır.
- InstructionBytes komutun saklandığı parametredir.
- Explanation komut ile ilgili açıklamayı saklayan parametredir.
- CheckSum byte dizisinin checksum değerini tutan parametredir.

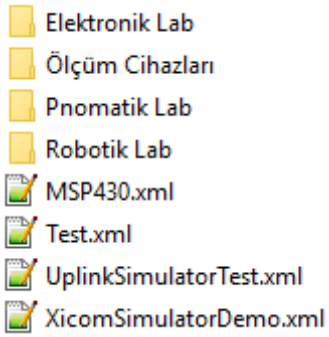
### 4.3. Cihaz Komut Dosyasının Yüklenmesi

Eğer daha önce ilgili cihaz için bir dosya oluşturulmamışsa Şekil 4.5'teki "File" menüsünde görülen "Create New Device XML File" kullanılarak boş bir yeni cihaz XML dosyası oluşturulur. Daha önce oluşturulmuş olan dosya "Open Device XML File" ile açılır. Sık sık yapılması gereken bu işlem için ana ekrandaki araç çubuğunda da Şekil 4.8'de görülen "Load Device File" butonu mevcuttur. Bir cihaz üzerinde çalışılması gerektiğinde ilgili cihazın komut listesini barındıran dosya bu şekilde yüklenir.




**Şekil 4.8** Bir cihaz dosyasının yüklenmesi

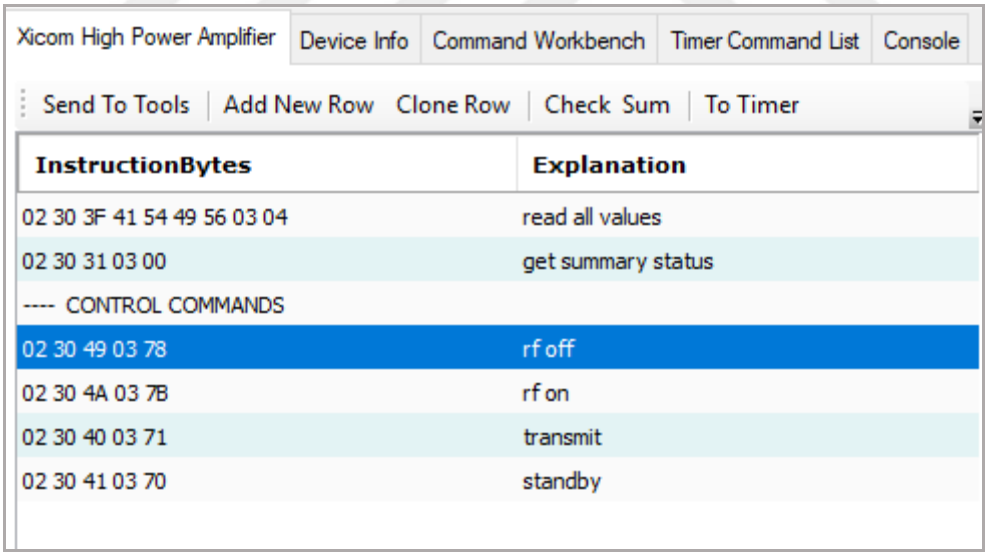
Yüklenen bu dosyalara gerektiğinde "devicexmls" klasörü açılarak da ulaşılabilir. Dosyalar açılarak üzerlerinde işlem yapılabilir veya dosyalar başkaları ile paylaşılabilir.




**Şekil 4.9** XML cihaz dosyalarının sistemdeki görünümü

Şekil 4.9'de "DeviceXMLs" klasörünün örnek bir görünümü mevcuttur. Belirli bir grup altında toplanabilecek cihaz dosyaları için bir klasör açıp, dosyaları o klasör içine kopyalayarak daha düzenli çalışmak mümkündür.

İlgili dosya seçilip "OK" butonuna basıldıktan sonra komut listesi bir grid üzerinde görüntülenir. Aynı zamanda "Command Workbench" sekmesine de komutlar metin şeklinde yüklenir. Komutlar üzerinde sürekli değişiklikler yapmak gerektiğinde bu kısmın kullanılması daha avantajlıdır.




**Şekil 4.10** Cihaz komut listesi görünümü

Şekil 4.10'da "Send to Tools" butonu seçili olan komutu, checksum hesaplayıcıya gönderir. Bu şekilde komut üzerinde bir değişiklik yapıldığında checksum hesaplanabilir. Checksum hesaplayıcı kısmında, yeni komut paketi oluşturulduğunda, tekrar burada göndermek üzere "Send to Grid" şeklinde bir buton mevcuttur.

Şekil 4.10'da "Add New Row" listeye yeni bir komut satırı ekler. "Clone Row" ise seçili olan komutun aynısından bir tane daha oluşturur. Bu komutta ufak bir değişlik yapmak gerektiğinde kullanılır.

"Check Sum" butonu, seçili olan komut paketinin "Current Checksum Type" kutusunda seçili olan checksum tipine göre checksum değerini hesaplar ve pakete ekler. Böylece "Checksum Calculator" kullanmaya gerek olmadan da checksum hesabı yapılabilir.

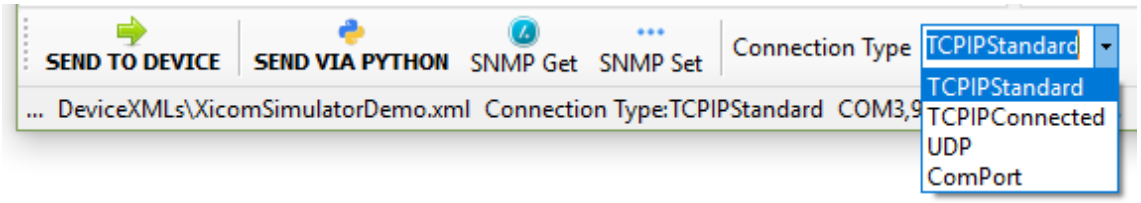
"To Timer" butonu seçiliş olan komutları "Timer Command List" sekmesindeki komut listesine ekler.

Sonuç olarak, programın Şekil 4.10'da görülen kısmı, komutların listelendiği ve üzerinde çalışıldığı kısımdır.

Bir sonraki aşama komutu cihaza göndermektir.

## 4.4. Komutların Gönderilmesi

Komutu cihaza göndermek için, gönderilecek olan komut seçilir. "**SEND TO DEVICE**" butonuna basıldığında komut cihaza o an seçili olan Şekil 4.11'de gösterilen bağlantı tiplerinden biri üzerinden gönderilir.




**Şekil 4.11** Komutu cihaza gönderme bölümü

Komutlar HEX formatta görsel olarak anlaşılması açısından tercihen aralarında boşluk bırakılarak yazılır. Örnek: "02 0D 1A 10 11 03 23". Boşluk bırakılmaması durumunda veya HEX olmayan karakterler de komutun içinde yer aldığında, boşluklar ve farklı karakterler program tarafından silinir.

Komutlar örneğin "GET_STATUS?" şeklinde ASCII formatta ise, komut gönderilmeden önce Şekil 4.12'de görülen "Command is ASCII" onay kutusu seçili olmalıdır.


**Şekil 4.12** Komut gönderimi ile ilgili parametreler

Program aslında bir "string" olan komut paketini byte dizisine çevirerek karşıya gönderir. Cevap alındıktan sonra HEX ve ASCII formatlarda gösterilir.

```
namespace DeviceTester
{
    public partial class frmMain : Form
    {
        byte[] GetBytes(string hexBytePacket)...

        //private void SendBytesToDevice(byte[] bytesToSend)
        private void SendBytesToDevice(string hexBytePacket)...

        private void SendViaSerialPort(byte[] bytesToSend)...

        private void SendViaTCPConnected(byte[] bytesToSend)...

        private void SendViaTCPStandard(byte[] bytesToSend)...

        private void SendViaUDP(byte[] bytesToSend)...

        private void SendViaPython(string bytesToSend)...

        SNMP METHODS
    }
}
```

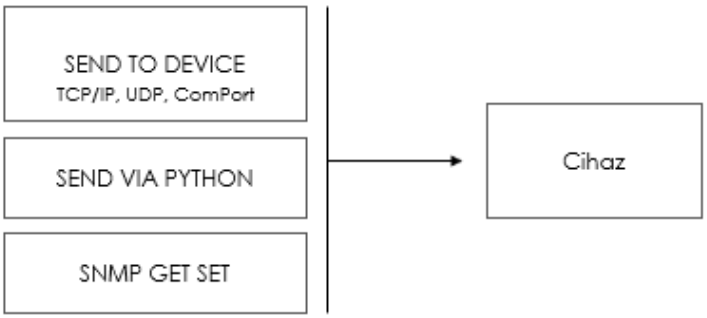
**Şekil 4.13** Komutların cihaza gönderildiği kod bölümü

"Save Log To Text File" seçili ise gönderme zamanı, gönderilen komut paketi, gelen cevap, bir metin dosyasına yazılır.

### 4.5. Bağlantı Tipleri

Mevcut bağlantı tipleri şu şekildedir:

- TCPIPStandard
- TCPIPConnected
- UDP
- SNMP
- Serial Port




**Şekil 4.14** Komut gönderme kanalları

Şekil 4.14'te birbirinden bağımsız yapıda bağlantı kanalları gösterilmiştir. Bu kanallar bağlantı tipinden farklı olup, kendi ilgili butonları üzerinden çalışır. Bundan dolayı programda Şekil 4.10'da görüldüğü gibi Python ve SNMP üzerinden veri göndermek için özel butonlar mevcuttur ve bir bağlantı tipi seçmeksizin bunlara tıklanarak seçili komut cihaza gönderilir.

Komut Şekil 4.11'de görülen "SEND TO DEVICE" butonu üzerinden gönderildiğinde gönderme işlemi bağlantı tipine göre gerçekleşir. Varsayılan bağlantı tipi "TCPIPStandard"tır. Bu bağlantı tipinde "SEND TO DEVICE" butonuna basıldığında otomatik olarak program tarafından bağlantı açılır, paket gönderilir ve ardından bağlantı kapatılır.

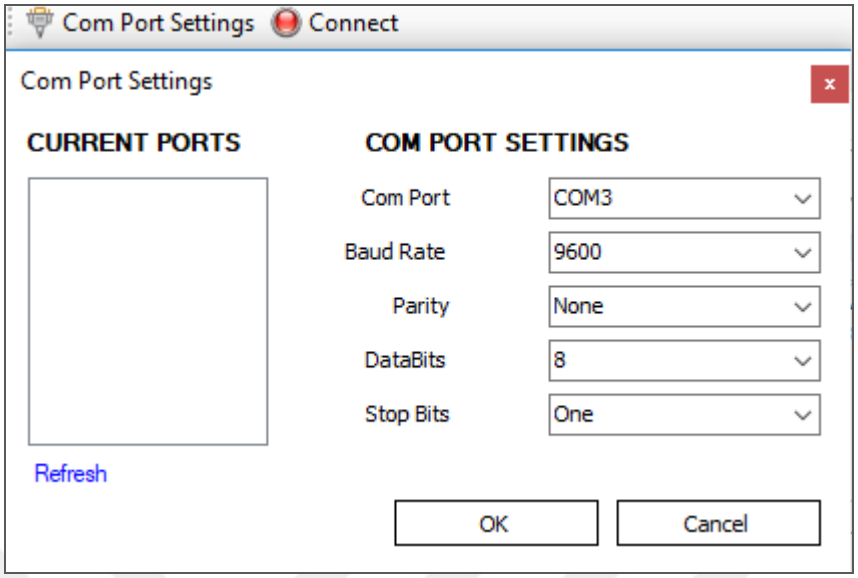
Bağlantının her komut gönderme işleminde açılıp kapatılmasının en büyük avantajı, bağlantı ile cihazı meşgul etmemektir. Program üzerinden bağlantı açıldığında, başka bir program üzerinden cihaza komut gönderilemez. Cihazlarla çalışma esnasında birden fazla programla cihaza komut gönderme durumu yaşanabilir. Bunlardan birisi örneğin cihazı üreten firma tarafından verilen cihazın kontrol programı olabilir. İkinci nokta, kullanıcının bağlanmayı düşünmeden çalışmasına olanak vermektir. IP ve port doğru olduktan sonra, artık bağlanma, bağlantıyı kesme şeklinde işlemlere gerek kalmadan çalışmak hız ve akışkanlık kazandırır.

Ancak gerekli olduğunda sürekli bağlı çalışabilmek için "TCPIPConnected" bağlantı tipi de mevcuttur. Bu seçildiğinde "Connect" butonu ile önce bağlantı yapılmalı, komut gönderme işlemleri bittikten sonra "Disconnect" yapılmalıdır.

Diğer bağlantı tipleri UDP ve COM Port'tur.

Seri haberleşme kullanılması gerektiğinde "Com Port Settings" butonuna basarak

açılan Şekil 4.15'teki pencerede bağlantı ayarları yapılmalıdır.




**Şekil 4.15** Seri haberleşme ayarları

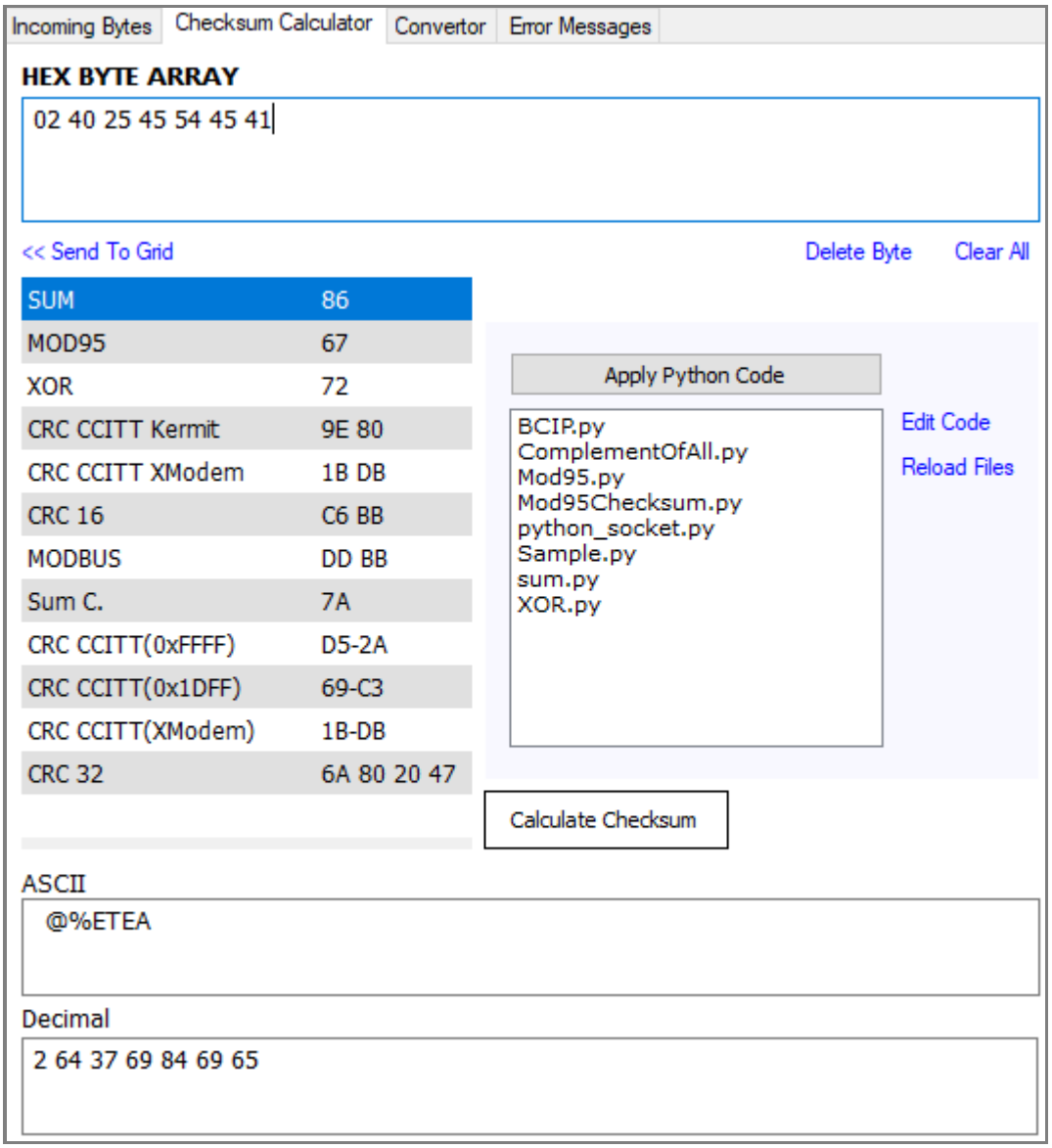
Burada yapılan ayarlar "Start.ini" dosyasında saklanmaktadır.

"Connect" butonuna tıklandığında bağlantı kurulur. Bu aşamadan sonra cihaza komut gönderilebilir.

UDP bağlantı tipinde de TCP/IP gibi sadece IP ve port gereklidir. Bunun dışında, UDP'nin yapısı gereği bağlantısız çalışır. Program üzerinden UDP bağlantılı çalışmak için yapılması gereken tek şey bağlantı tipi olarak UDP'yi seçmektir.

### 4.6. Komutun Oluşturulması

Byte seviyesinde komutlarla çalışılması gerektiğinde ve komutun yapısında checksum bulunduğunda, programın "Checksum Calculator" kısmı kullanılır.



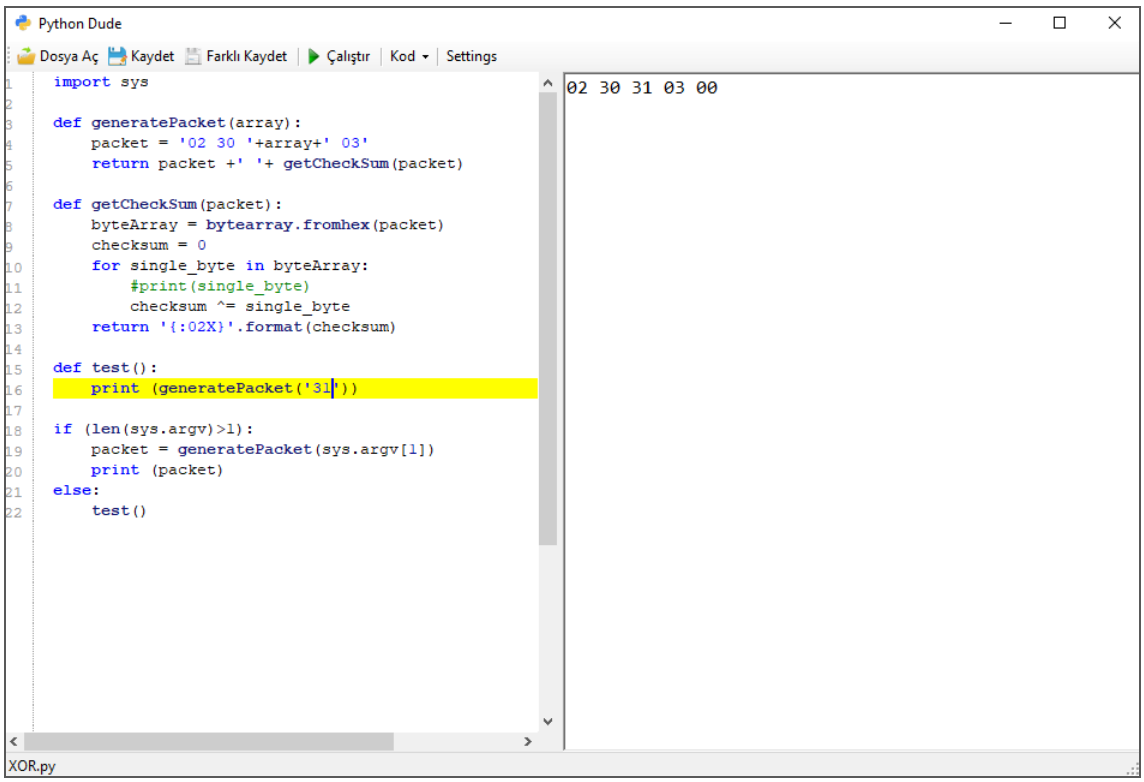

**Şekil 4.16** Komut paketi oluşturma arayüzü

Şekil 4.16'da görülen "HEX BYTE ARRAY" kısmında byte değerleri heksadesimal olarak yazıldığında, aynı anda ASCII ve desimal karşılıkları da görülür.

Byte değerlerinin yazılması esnasında, otomatik olarak programın içinde mevcut checksum tiplerine göre program mevcut byte dizisinin checksum değerlerini hesaplar ve listeler. Listede çift tıklayınca checksum yukarıdaki byte dizisine eklenir ve daha sonra "Send To Grid" butonuna basarak, oluşan komut sol taraftaki ana komut listesinde o anda seçili olan satıra kaydedilir.

### 4.7. Komut Oluşturmada Python Desteği

Eğer listedeki checksum tipleri yeterli değilse, kullanıcı Python kodu yazarak kendi istediği checksum hesabını yaptırabilir. Bu durumda kullanıcının yapması gereken listede ilgili Python dosyasını seçerek "Apply Python Code" butonuna basmaktır. Bu yapıldığında, byte dizisi Python'a gönderilir. Python aldığı byte dizisi üzerinde hesaplamayı yapar ve sonucu döndürür. Program tarafından alınan sonuç "HEX BYTE ARRAY" kısmında gösterilir.

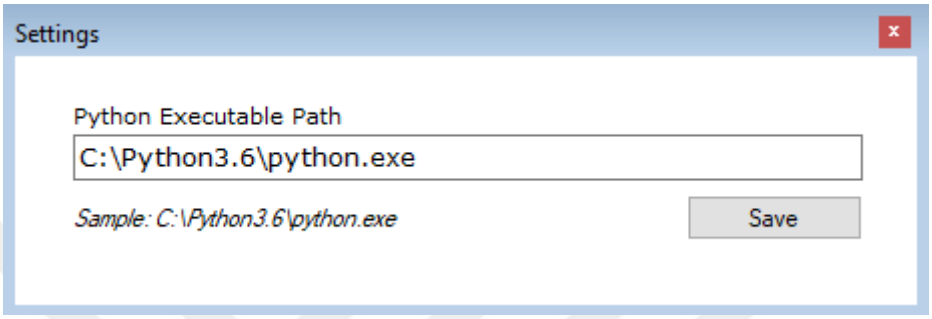



**Şekil 4.17** Python kodlama arayüzü

Bunun ötesinde, sadece checksum hesaplamanın dışında, komutun oluşturulması için gerekli başına 0x02, sonra cihazın adresi, sonra komut byteları, sonra 0x03 ve 0x03'ten sonra checksum koymak gibi rutin işleri de Python yapabilir. Şekil 4.17'deki örnek bunu göstermektedir. Örnekte "generatePacket" fonksiyonu girilen byte dizisinin başına 0x02 koyuyor, burada 0x30 adresli bir cihaz mevcut, "array" değişkenine gönderilen komut (0x31) atanıyor ve sonra 03 diziye ekleniyor ve son olarak "getChecksum" fonkiyonu ile tüm paketin checksum hesabı yapılıp pakete eklenerek döndürülüyor.

Bu kullanıcıya zaman kazandırıyor ve hesaplamalar esnasında hata yapma ihtimalini azaltıyor.

Python'un program üzerinden kullanılabilmesi için, Python sistemde kurulu olmak zorundadır. Python kodu yazabilmek veya program üzerinde daha önceden yazılmış Python kodlarını görebilmek için, bu tezde geliştirilen programa için yazılmış, basit bir Python editör programı mevcuttur. Bu editör Şekil 4.17'de görülmektedir.



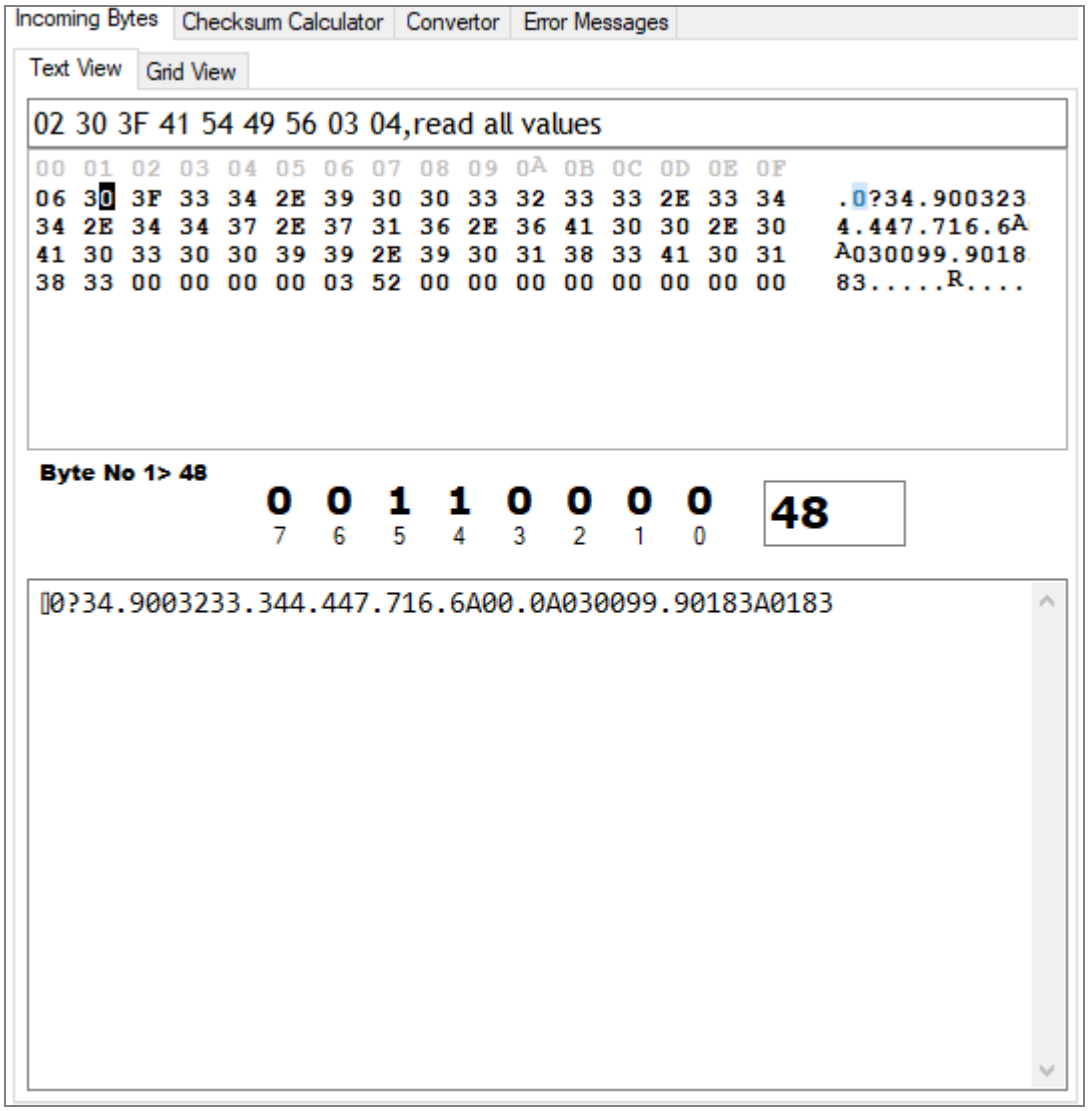

**Şekil 4.18** Python.exe'nin yolunun Python Editor için tanımlanması

Şekil 4.18'deki pencerede Python editörün çalışabilmesi için Python.exe dosyasının yolu tanımlanmalıdır.

PythonDude programı "Çalıştır" butonuna basıldığında, Şekil 4.16'daki pencerenin sol tarafında görülen kodu, tanımlı klasördeki python.exe dosyasına gönderir. Python.exe tarafından çalıştırılan kodun geri döndürdüğü değer Şekil 4.16'da gösterildiği gibi pencerenin sol tarafında görüntülenir.

Python kodunun DeviceTester üzerindeki çalışma şekli ise şu şekildedir. DeviceTester üzerinde Şekil 4.16'da görülen "Apply Python Code" butonuna basıldığında, butonun alt tarafındaki listede seçili olan python dosyasındaki kod çalıştırılır. Python.exe tarafından çalıştırılan koddan geri dönen değer "HEX BYTE ARRAY" kutusunda gösterilir.

### 4.8. Gelen Cevabın Görüntülenmesi ve Analizi



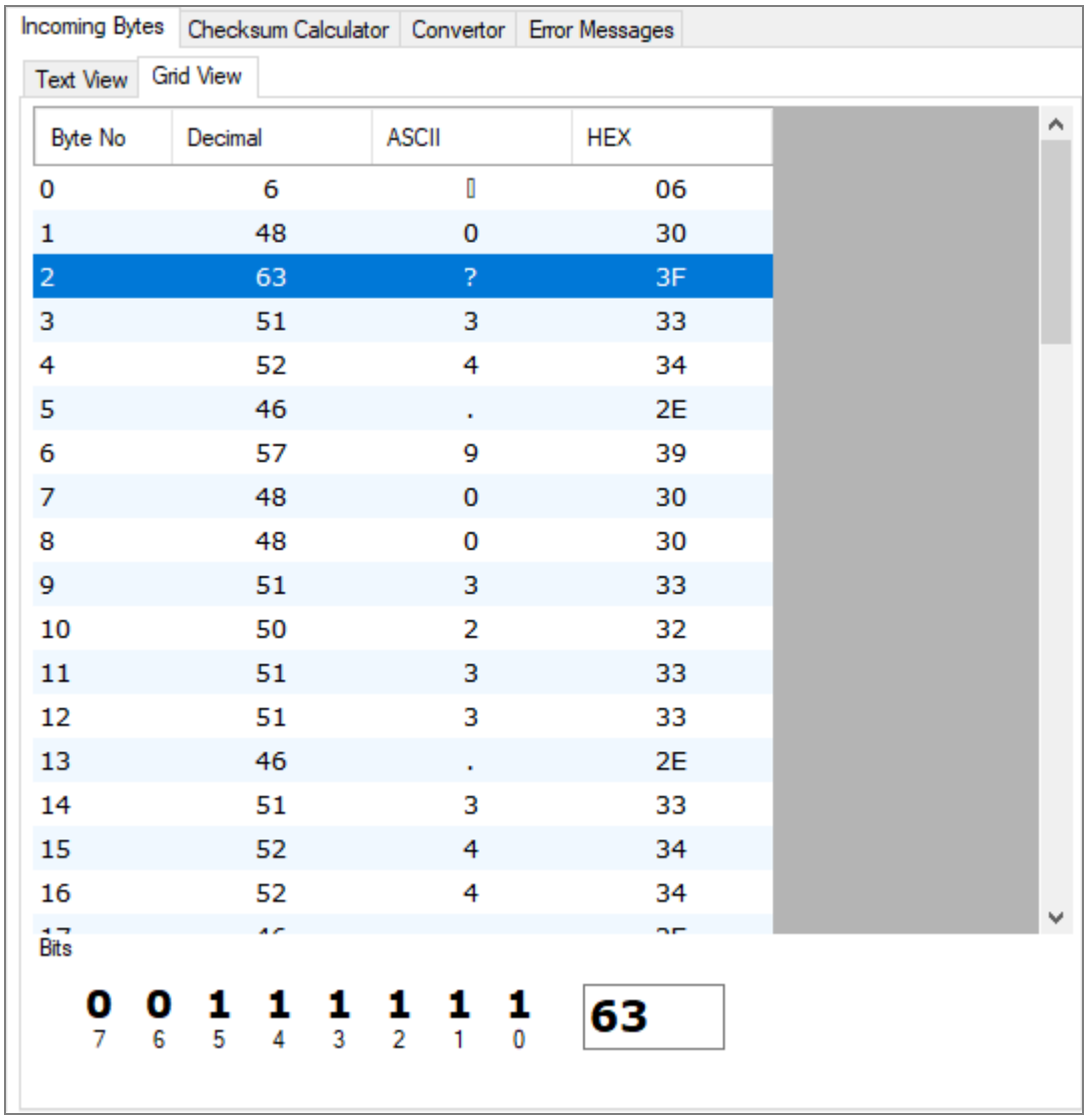

**Şekil 4.19** Gelen cevabın görüntülendiği ana arayüz

Şekil 4.19'da görülen cihazdan gelen cevabın görüntülendiği arayüzün en üst kısmında gönderilen komut ve açıklaması yer alır. Bu kısmın altında, düşük seviyeli sistemlerde sık kullanılan bir gösterim şekli mevcuttur. Bu gösterimde en üst kısımda byte numaraları, alt sol kısımda verinin HEX formatta ve sağ kısımda ASCII formatta gösterimi mevcuttur.

Herhangi bir byte değeri üzerine tıklandığında, orta kısımda bu değerin bit seviyesinde açılımı ve desimal değeri görüntülenir. Bu kullanıcıya hızlı bir şekilde her byte değerinin bit açılımını görme imkanı verir. Bu kısımda ayrıca üzerine çift tıklanarak bit değerleri değiştirilebilmekte ve girilen bir desimal değerin bit değerleri

görüntülenmektedir. Bu tip özellikler, örneğin farklı bir değer beklendiğinde, bu byte değeri için gelen cevabın kaçıncı biti 1 veya 0 olmalıydı gibi sorulara hızlı bir şekilde cevap vermeyi sağlar.

ASCII formatta gösterimin daha net görülebilmesi için, ayrıca en alt kısımda da daha geniş bir şekilde ASCII gösterim yer almaktadır.




**Şekil 4.20** Gelen cevabın grid içerisindeki görünümü

Gelen cevap üzerine analiz yaparken, özellikle alınan byte dizisinin herhangi bir elemanının kaçıncı byte olduğunu bilmek gerekebilir. Bu sorunun cevabı Şekil 4.20'de görülen grid üzerindeki gösterimde mevcuttur. Herhangi bir byte değerinin kaçıncı byte olduğu bilgisinin yanında, decimal, ASCII, HEX değerlerin tamamı tek bakışta görülebilmektedir.

Herhangi bir byte değerinin bit değerlerinin görüntülenmesi özelliği burada da mevcuttur.

Tüm bu özellikler gelen cevap üzerinde herhangi farklı bir yazılıma gerek duymadan, hızlı bir şekilde yorum yapmayı mümkün kılar.

### 4.9. Zamanlayıcı Kullanarak Komut Çalıştırma

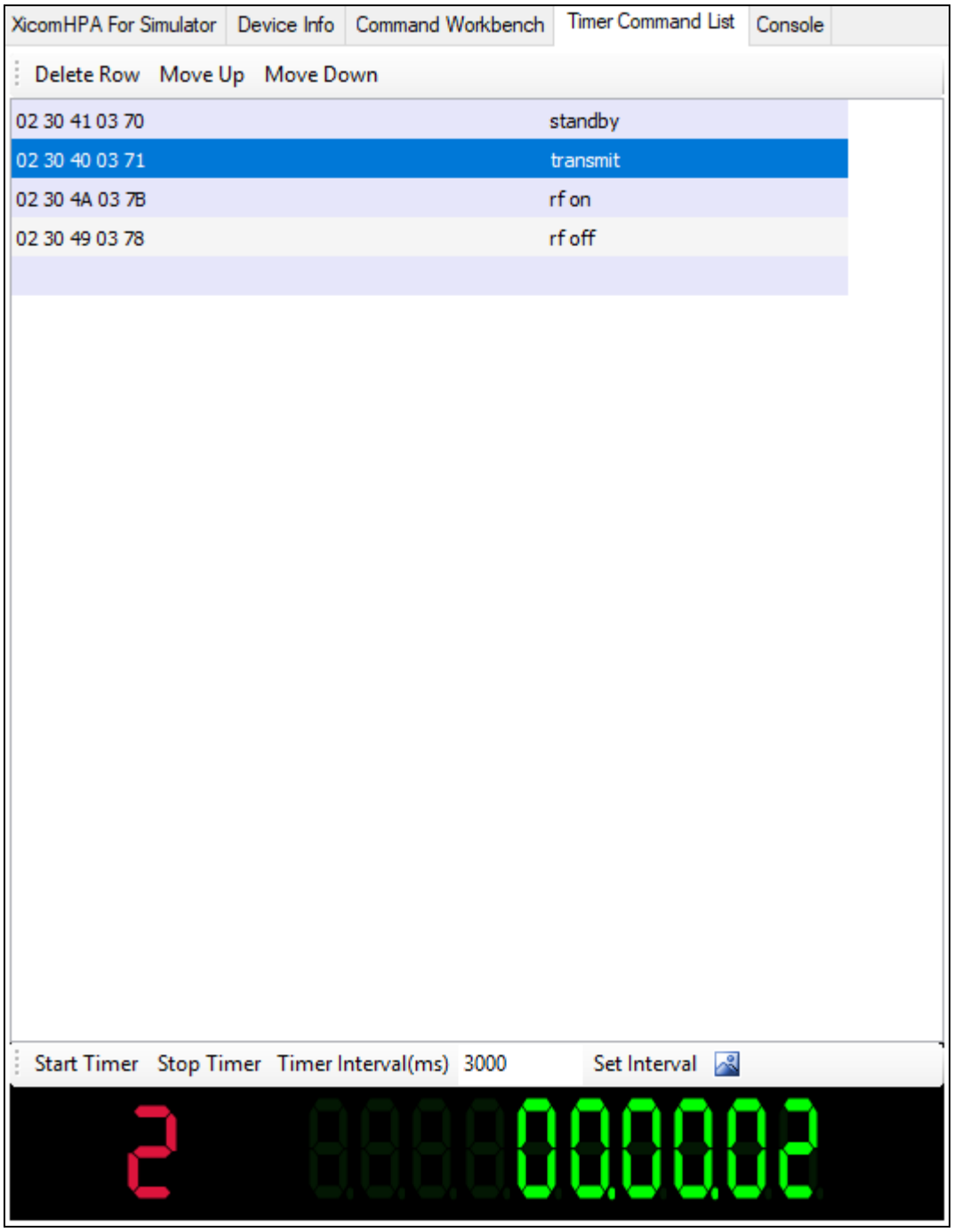
Bir zamanlayıcıya bağlı olarak, istenen bir dizi komutun belirli zaman aralıklarında çalıştırılması ihtiyaç duyulan bir özelliktir.

Komutların bir dosyada liste halinde saklanması, bu özelliğin programa rahat bir şekilde eklenmesini sağlamıştır. HyperTerminal, Putty gibi diğer programlarda komutları saklama özelliği olmadığından, bir listede sıralanmış komutları belirli zaman aralıklarında cihaza gönderme özelliği o programlar için söz konusu değildir.

Şekil 4.10'da görülen komut listesinde seçilen komutlar, Şekil 4.21'de görülen "Timer Command List" sekmesinde bulunan gride "To Timer" butonuna basılarak eklenir.

Şekil 4.21'de görülen "Start Timer" butonuna basıldığında zamanlayıcı çalışmaya başlar ve listedeki komutları "Timer Interval" kutusunda milisaniye olarak belirtilen zaman aralıklarında cihaza sırayla gönderir. Gönderme işlemini durdurmak için "Stop Timer" butonuna basılır.

Gönderme işlemi devam ederken farklı bir zaman aralığı değeri girilip "Set" butonuna basılarak işlem devam ederken de zaman aralığı değeri değiştirilebilir.

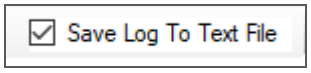



**Şekil 4.21** Zamanlayıcı komut listesi görünümü

Şekil 4.21'de ekranın alt kısmında sol tarafta kırmızı renkle gösterilen değer, o anda gönderilen komutun listedeki sıra numarasını göstermektedir. Sol tarafta yeşil renkle gösterilen değer ise zamanı göstermektedir. Şekil 4.21'de şu an 2.komut için 2 saniye geçmiştir ve zaman aralığı 3 saniye olduğu için 1 saniye sonra komut cihaza gönderilecektir.

## 4.10. Gelen cevabın kaydedilmesi

Şekil 4.22'de görülen onay kutusu işaretli ise gönderilen ve alınan veriler loglanır.




**Şekil 4.22** Log kaydetme seçimi

Varsayılan değer olarak program, bütün gönderilen ve gelen cevapları "Logs" klasörü içerisinde oluşturulan "cihaz ismi+logs.xml" ismindeki bir XML dosyasına kaydeder.

| LOG TIME | INSTRUCTION | DATA IN HEX | DATA IN ASCII |
|---|---|---|---|
| 6/21/2018 3:39:56 PM | 02 30 31 03 00,get summary status | 6 48 49 8 0 0 3 12 0 0 0 0 0 0 0 0 0 0 0 0 | 01 |
| 6/21/2018 3:40:00 PM | 02 30 49 03 78,rf off | 6 48 73 73 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0II5 |
| 6/21/2018 3:40:02 PM | 02 30 4A 03 7B,rf on | 6 48 74 74 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0JJ5 |
| 6/21/2018 3:40:03 PM | 02 30 40 03 71,transmit | 6 48 64 64 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0@@5 |
| 6/21/2018 3:40:05 PM | 02 30 41 03 70,standby | 6 48 65 65 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0AA5 |
| 6/21/2018 3:40:07 PM | 02 30 3F 41 54 49 56 03 04,read all values | 6 48 63 51 52 46 57 48 48 51 50 51 51 46 51 52 52 46 52 52 | 0?34.9003233.344.44 |
| 6/21/2018 3:41:11 PM | 02 30 3F 41 54 49 56 03 04,read all values | 6 48 63 49 52 46 57 48 48 51 50 50 49 46 50 51 52 46 52 52 | 0?14.9003221.234.44 |
| 6/21/2018 3:41:35 PM | 02 30 31 03 00,get summary status | 6 48 49 11 0 0 3 15 0 0 0 0 0 0 0 0 0 0 0 0 | 01 |
| 6/21/2018 3:41:36 PM | 02 30 31 03 00,get summary status | 6 48 49 11 0 0 3 15 0 0 0 0 0 0 0 0 0 0 0 0 | 01 |
| 6/21/2018 3:41:38 PM | 02 30 49 03 78,rf off | 6 48 73 73 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0II5 |
| 6/21/2018 3:41:41 PM | 02 30 40 03 71,transmit | 6 48 64 64 3 53 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0@@5 |

**Şekil 4.23** Örnek bir log dosyası görünümü

Log dosyasında aşağıdaki parametreler XML etiketleri halinde kaydedilir:

- Tarih
- Gönderilen komut ve açıklaması
- Gelen cevap HEX
- Gelen Cevap ASCII

Şekil 4.23'te görülen log dosyasında, komutun cihaza gönderildiği tarih ve saat, açıklaması ile birlikte gönderilen komut, gelen cevabın HEX formatta ve ASCII formatta görünümleri farklı renklerdeki sütunlarda görüntülenmektedir.

## 4.11. Yapılan Çalışmanın Özelliklerinin Diğer Programlarla Karşılaştırması

Tablo 4.1'de bu tezde yapılan çalışmanın diğer programlarla temel özellikler konusunda kıyası yapılmıştır.

Tablo 4.1 DeviceTester'ın Diğer Programlarla Kıyası

| | DeviceTester | HyperTerminal | Putty | PacketSender | Hercules |
|---|---|---|---|---|---|
| Komutları düzenleme ve kaydetme | + | | | | |
| Gelen cevabı loglama | + | + | + | + | |
| Gelen cevabı ASCII, Hex gösterme | + | | | + | |
| Checksum hesaplayıcı | + | | | | |
| Zamana bağlı ardarda komut gönderme | + | | | | |
| "Byte to bit" dönüştürücü | + | | | | |
| TCP/IP Haberleşme | + | + | + | + | + |
| UDP Haberleşme | + | | + | + | + |
| Seri Haberleşme (COM PORT) | + | + | + | | + |

Tablo 4.1 giriş kısmında belirtilen, bir cihaza komut gönderme ve gelen cevabı görüntüleme ve inceleme sürecinde gerekli olan en temel özellikleri içerir. Bu tezde geliştirilen uygulamaya bu özelliklerin tamamı eklenmiştir ve bu özelliklerin gelen cevabın loglanması dışındaki hiçbirinin diğer programlarda olmadığı görülmektedir.

HyperTerminal ve Putty programındaki loglama da mevcut yapılan işlemlerin saklanmasından ibarettir.

Bu programlar cihazla haberleşme için de kullanılmalarına rağmen, temelde daha farklı amaçlar için geliştirilmişlerdir. DeviceTester tamamen cihazla haberleşme temel amacına göre geliştirilmiş bir uygulamadır.

## 4.12. Örnek Bir Uygulama

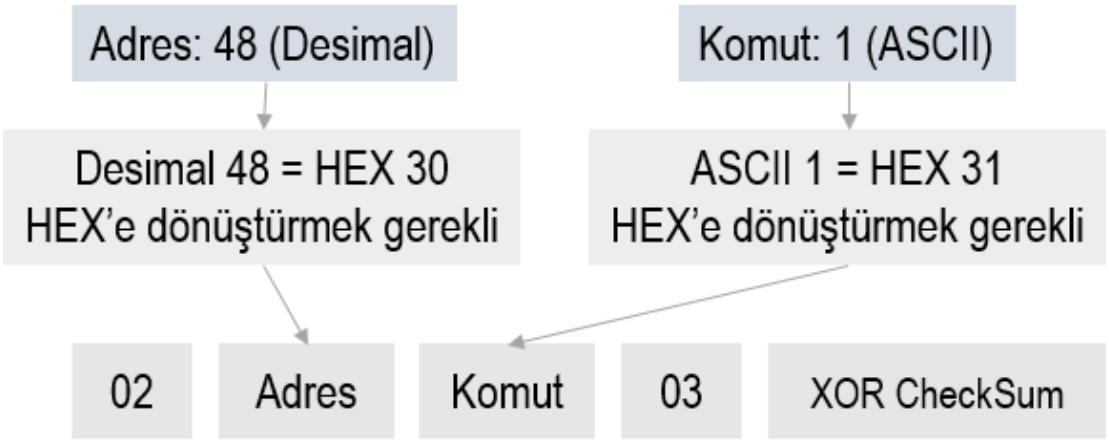
Bu uygulamada gerçek bir cihazın komutlarının test edilmesinde bu çalışmada geliştirilen DeviceTester olmadan ve DeviceTester kullanılarak sürecin işleyişi incelenecek ve DeviceTester'ın kazanımları gösterilecektir. Cihaz bir yüksek güç amplifikatörüdür.

### 4.12.1 DeviceTester Olmadan Komut Paketi Oluşturma

Cihazın komut protokolü Şekil 2.3'te gösterilen STX-ETX protokolüdür. Buna göre komut paketinin başlangıç değeri 02'dir. Bir sonraki byte cihazın adres değeri, sonraki byte komut, paketin bitiş byte değeri 03 ve sonraki byte ise 02 ve 03 dahil, bunların arasındaki tüm byte değerlerinden elde edilen XOR checksum değeridir.

Cihazın adresi desimal olarak 48'dir. Kullanılacak komut desimal olarak 49'dur. Bu komut cihazın alarm durumu, çıkış gücünün aktifliği gibi parametreleri ile ilgili mevcut durumu sorgulamaktadır.

Burada dikkat edilmesi gereken bir nokta, bu verilerin ASCII veya HEX olarak da verilebileceğidir. Örneğin 49 komut değerinin ASCII karşılığı 1'dir ve cihazın komutlarının açıklandığı katalogda ASCII olarak belirtilmiştir. Ancak bu protokol HEX format kullanılarak çalışılan bir protokoldür.




**Şekil 4.24** STX-ETX protokolüne göre paketin oluşturulması

Bu bilgiler doğrultusunda Şekil 4.24'te görüldüğü şekilde, adım adım komut paketi oluşturulacaktır. HEX formatta çalışmak gerektiğinden 49 desimal değerinin heksadesimal karşılığı gereklidir. Bunun için Windows hesap makinasının

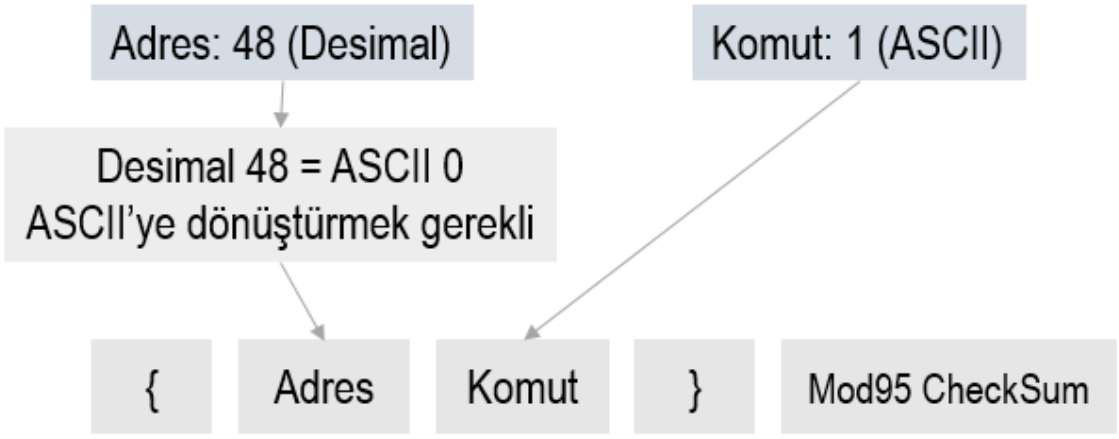
programlayıcı kısmı kullanılabilir. Bu şekilde veya başka yollarla 49 desimal değerinin heksadesimal karşılığı 30 bulunur. Sonraki adım, cihazın durum bilgisini veren 1 ASCII komut değerinin HEX karşılığını bulmaktır. Bunun için bir yöntem ASCII karakter tablosundan 1'in HEX karşılığına bakmaktır.

Son adım XOR checksum hesabıdır. Bunu yapmak için, internet üzerinde checksum hesabı yapan siteler kullanılabilir. Veya kullanıcı kendisi bu hesabı yapan bir kod yazmalıdır. Bu örnek için XOR checksum hesabı yapıldığında sonuç 00 bulunur. Buna göre komut paketi "02 30 31 03 00" dır.

Bu somut örnekte, üç işlem için farklı araçlara ihtiyaç duyuldu. Bu işlemlerin birincisi desimal adres değerini HEX formata dönüştürme, ikincisi ASCII verilen komut değerini HEX formata dönüştürme ve üçüncüsü XOR checksum hesaplamadır.

Aynı cihazın 2, 3, 4 gibi farklı komutları için bu dönüşüm ve hesapların yeniden yapılması gerekir. Ayrıca farklı bir adreste ikinci bir cihazla daha çalışılması gerekiyorsa, bu durumda sadece adres değişiminden dolayı tüm paketlerin yeniden hesaplanması gerekir.

Yine aynı cihazda Şekil 4.25'te görülen benzer yapıda ASCII bir protokol seçeneği de mevcuttur. Kullanıcı bu seçeneği de kullanmak isteyebilir. Bu durumda farklı bir checksum hesabı gereklidir.

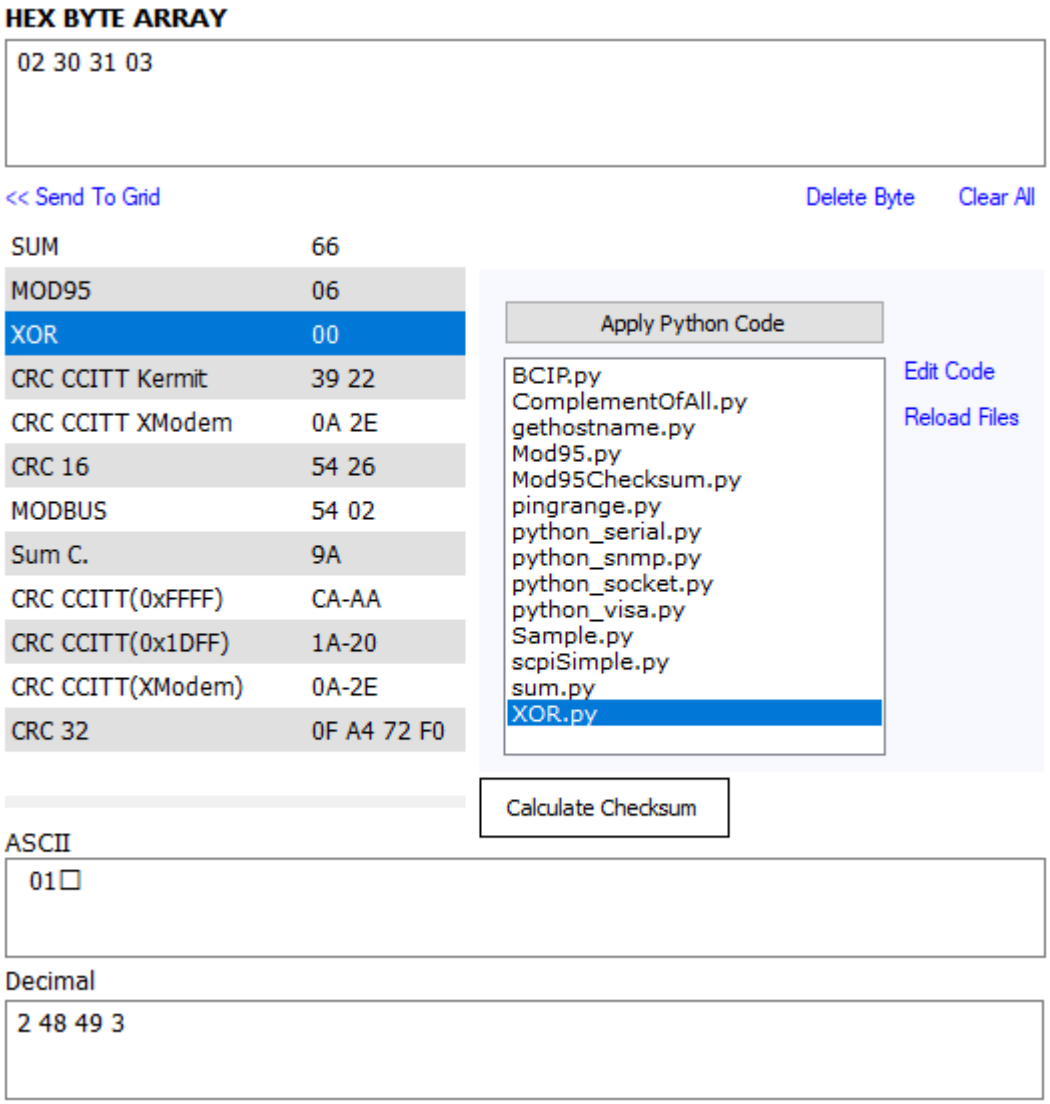



**Şekil 4.25** ASCII protokolde komut paketinin oluşturulması

Paket oluşturma işlemi bittikten sonra kullanıcı oluşturduğu paketi, Hercules veya PacketSender gibi bir programla cihaza gönderir.

### 4.12.2 DeviceTester Kullanılarak Komut Paketi Oluşturma

Aynı işlemler DeviceTester üzerinde şu şekilde gerçekleşir. HEX BYTE ARRAY kutusunda 02 yazıldıktan sonra Şekil 4.26'da alt kısımda görülen Decimal kutusunda 48 yazılır. Otomatik olarak HEX BYTE ARRAY kutusuna 48'in heksadesimal karşılığı 30 eklenecektir. Sonraki adım ASCII verilen 1 komutunu ASCII kutusuna yazmaktır. Bu değer de HEX BYTE ARRAY kutusuna ASCII 1'in HEX karşılığı 31 olarak eklenir. Sonraki adım paketin sonuna 03 eklemektir. 03 yazıldığında otomatik oluşan listede XOR checksum değeri görülmektedir. Üzerine çift tıklanarak pakete eklenir ve "Send To Grid" butonuna basılarak komut listesine eklenir.




**Şekil 4.26** DeviceTester üzerinde komut paketinin oluşturulması

Bütün hesaplamalar programın içinde mevcuttur. Yapılması gereken sadece değerleri paket içerisinde gerekli yerlere yazmaktan ibarettir.

Python desteği ile bu hızlandırma bir adım daha ileriye taşınmıştır.

```
import sys

def generatePacket(array):
    packet = '02 30 '+array+' 03'
    return packet +' '+ getCheckSum(packet)

def getCheckSum(packet):
    byteArray = bytearray.fromhex(packet)
    checksum = 0
    for single_byte in byteArray:
        checksum ^= single_byte
    return '{:02X}'.format(checksum)

def test():
    print (generatePacket('0A 0B 0C'))

if (len(sys.argv)>1):
    packet = generatePacket(sys.argv[1])
    print (packet)
else:
    test()
```

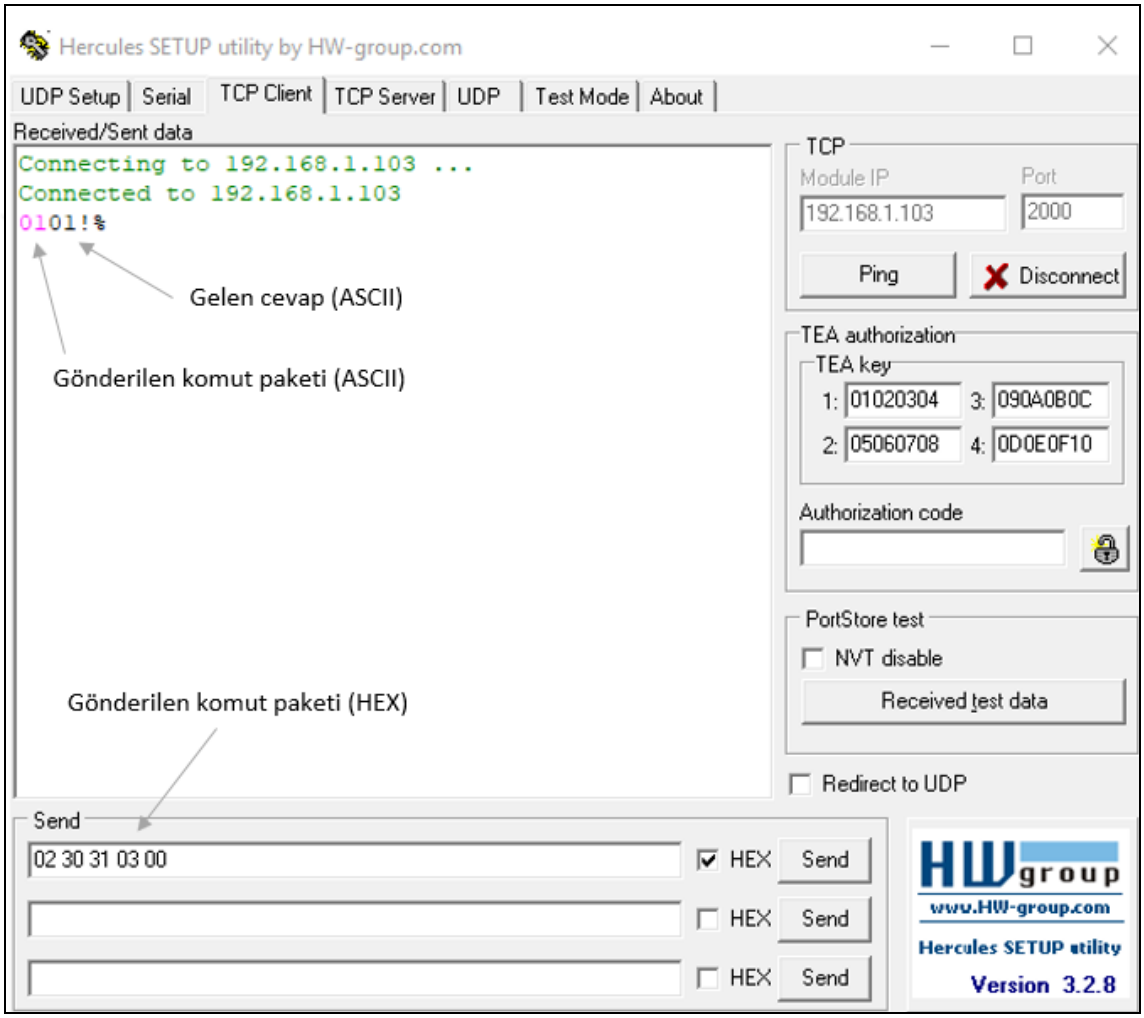
**Şekil 4.27** Python ile STX-ETX protokolünün uygulanması

Python üzerinden komut paketi oluşturmak için sadece Şekil 4.25'teki HEX BYTE ARRAY içerisine 31 yazıp, sağ taraftaki Python dosyaları listesinde XOR.py seçili iken "Apply Python Code" butonuna basmak yeterlidir. Butona basıldığında 31 değeri Şekil 4.26'da görülen generatePacket(sys.argv[1]) kısmında sys.argv[1]'in değeri olarak alınır ve generatePacket('31') şeklinde çalıştırılır. Kodun üst kısmında generatePacket fonksiyonuna bakıldığında, 31 komut değerinin 02, adres değeri 30 ve 03 ile protokole uygun bir şekilde sarmalandığı ve bir adım sonra checksum değerinin hesaplanarak pakete eklenip paketin geri döndürüldüğü görülür. Cihazın adresinin değişmesi durumunda, kullanıcının buradaki 30 değerini değiştirmesi yeterlidir.

Python kodundan gönderilen paket, HEX BYTE ARRAY içerisinde görüntülenir. Bu şekilde DeviceTester üzerinde paket oluşturmak, sadece komutun kendisini yazıp butona basmaya indirgenmiştir.

### 4.12.2 DeviceTester Olmadan Komut Paketinin Gönderilmesi ve Gelen Cevabın Görüntülenmesi

Komut paketinin oluşturulmasından sonra, cihaza göndermek üzere bir program gereklidir. Kullanıcı araştırma yaparak uygun bir program bulmak durumundadır. Burada verinin farklı formatta görünümü ile ilgili temel bir problemi göstermek üzere önce Hercules, daha sonra buna çözüm getiren PacketSender kullanılacaktır.




**Şekil 4.28** Hercules üzerinden cihaza komut paketi gönderme

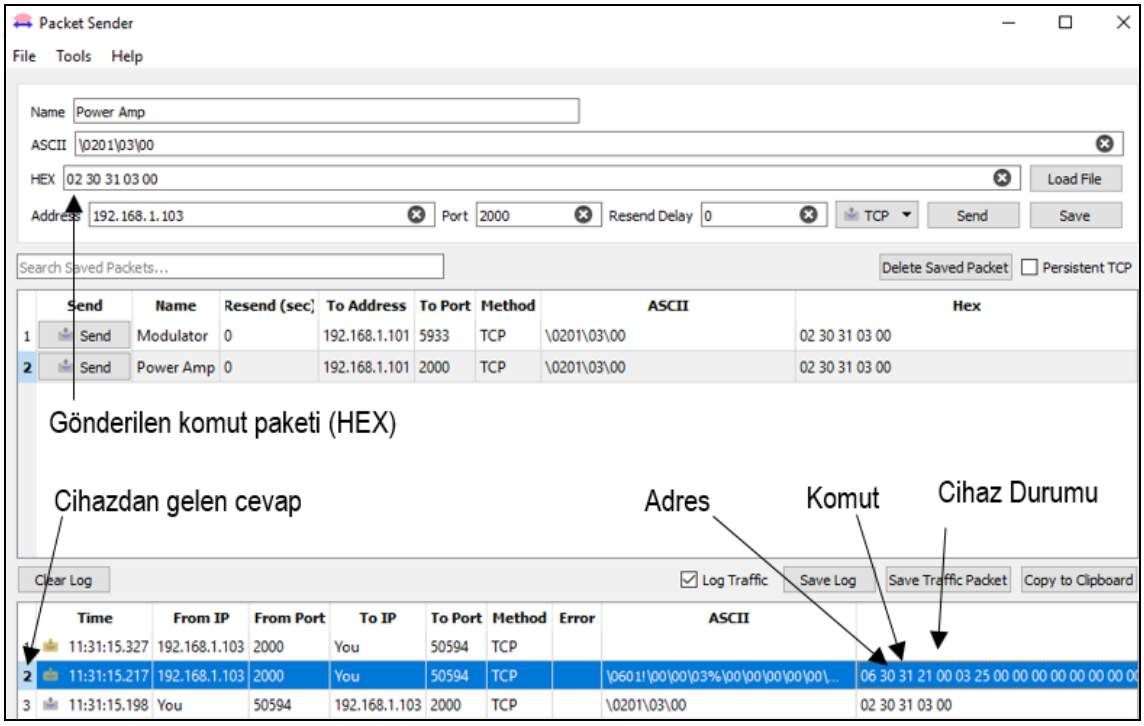
Şekil 4.28'de IP ve port değerleri yazılıp cihaza bağlandıktan sonra, oluşturulan paket ilgili kutucuğa yazılmış ve paketin yanındaki onay kutusunda HEX olduğu belirtilmiştir. Bu onay kutusu seçilmezse program veriyi ASCII kabul eder.

"Send" butonuna basılarak paket cihaza gönderilmiştir. Şekil 4.28'de görüldüğü gibi, gelen cevap bu programda ASCII formatta gösterilmektedir. Kırmızı renkli kısım,

gönderilen komuttur. Siyah renkli kısım gelen cevaptır. HEX formatta çalışılan bir sistemde, ASCII gösterimle gelen cevabı kesin olarak yorumlamak imkansızdır. Örneğin gelen cevapta ASCII olarak 0'ın cihazın adres değeri, 1'in gönderilen komut olduğu zaten biliniyor, ancak gelen cevapta ASCII karşılığı ekranda görüntülenemeyen bir değer varsa, cevabın yorumlaması yanlış olacaktır.

Bu durumda Hercules HEX formatta gösterim imkanı vermediği için başka bir program kullanılmalıdır.

Aynı örneğin PacketSender programı üzerindeki uygulaması Şekil 4.29'da gösterilmiştir.




**Şekil 4.29** PacketSender üzerinden cihaza komut paketi gönderme

Şekil 4.29'da görüldüğü gibi PacketSender üzerinden komut paketi gönderilmiş ve program cevabı HEX formatta göstermiştir.

Sonraki aşamada "06 30 31 21 00 03 25" şeklinde gelen cevap paketinin yorumlanması gereklidir.

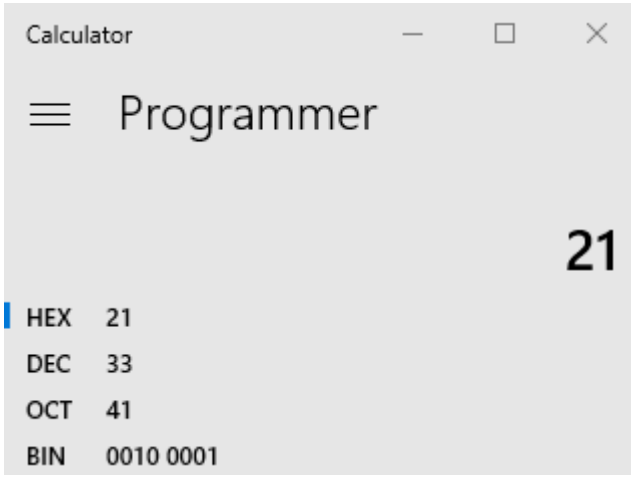
Cevapta STX-ETX protokolü gereği 0x06 gelen cevabın ilk byte değeridir. 0x30 adres, 0x31 gönderilen komuttur. Sonraki 2 byte durum bilgisi, 0x03 paketi sonlandırma byte değeri, 0x25 ise paketin XOR checksum değeridir.

| Format | Response | | | |
|---|---|---|---|---|
| 1 | AB | Byte A | bit 0 | **RF Inhibited** |
| | | | 1 | **Constant Power** |
| | | | 2 | **Remote** |
| | | | 3 | **Standby** |
| | | | 4 | **HV ON**<br>Transmit ON *(SSPA)* |
| | | | 5 | **Supply Enabled** *(via controller)*<br>or manual mode *(KPA)* |
| | | Byte B | bit 0 | **HV pending** (selected but undetected) |
| | | | 1 | **HV reduced** *(see command N)*<br>or power save *(KPA)*<br>or amp not connected *(via XTC-100D)* |
| | | | 2 | **Alarm** (send 9 for more info) |
| | | | 3 | **Power Changed** at least 0.2 dB since last power query |
| | | | 4 | **Heater Standby**<br>Auto Heater *(KPA)* |
| | | | 5 | **Summary Fault** (send 3 for more info) |

**Şekil 4.30** Güç amplifikatörünün durum bilgisinin bit seviyesinde açıklaması [27]

Bu aşamada cihazdan gelen paketin içerisindeki bu iki adet byte değeri Şekil 4.30'da gösterilen tabloya göre yorumlanmak durumundadır.

Her byte değeri bit şeklinde açılmalıdır. Durum bilgisinin Şekil 4.30'da Byte A olarak belirtilen birinci byte değeri, 0x21'in bit gösterimi elde edilerek Şekil 4.30'daki tabloya göre değerlendirilmelidir. Buna göre Şekil 4.31'de görüldüğü gibi Windows hesap makinası programlayıcı modda kullanılarak, 0x21 heksadesimal sayısının bit görünümü elde edilir.

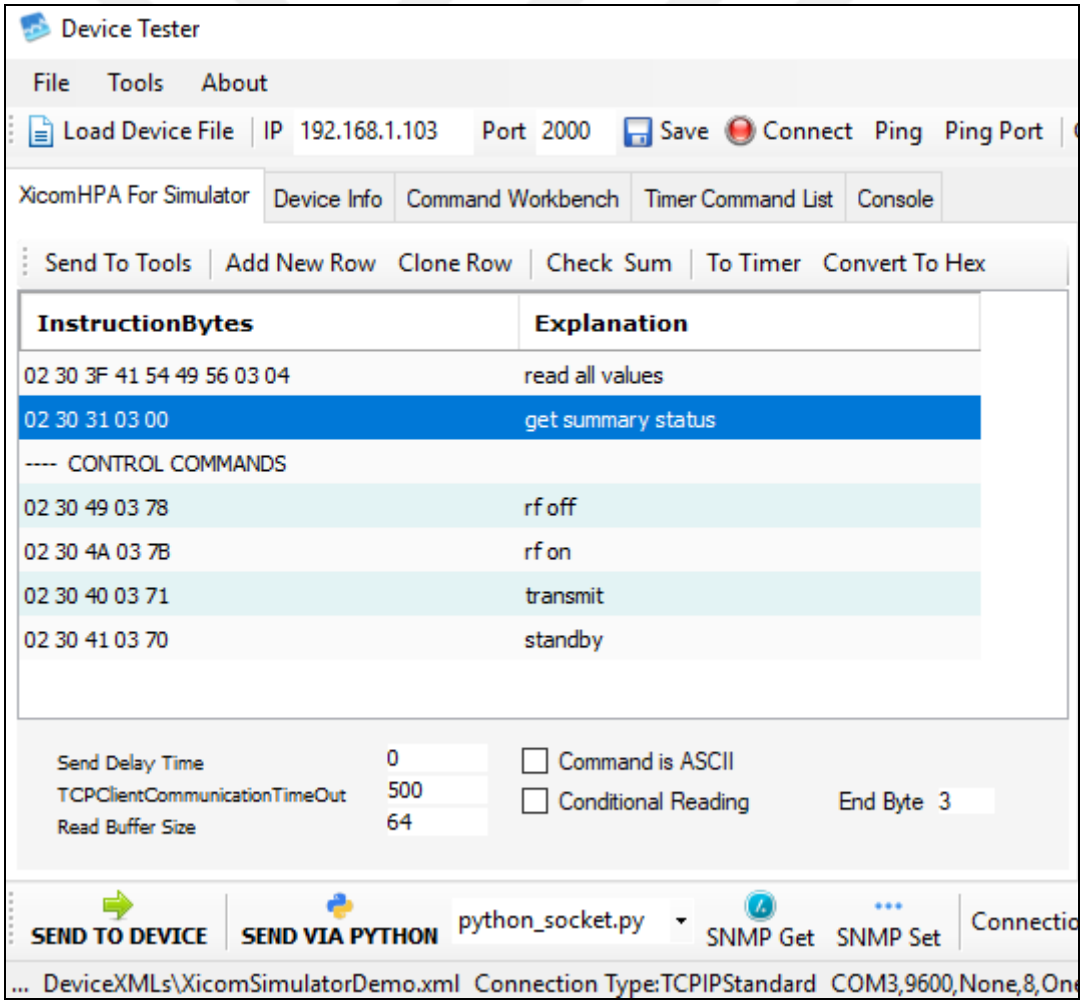



**Şekil 4.31** Windows hesap makinasında HEX bir değerin bit görünümü

Elde edilen bit değeri “0010 0001” dir. Şekil 4.30’daki tabloya göre, bit 0 = 1 olması durumunda cihaz “RF Inhibited” konumdadır. Ayrıca gelen değerde bit 5=1 olduğu görülmektedir. Bu cihazda “Supply Enabled=1” olduğunu, yani “Supply Enabled” özelliğinin aktif olduğunu gösterir. Kullanıcı bu şekilde, cihazın kataloğundaki tablolarda bit seviyesinde açıklaması yapılmış durum bilgisini incelemek için, her bir byte değerinin bit açılımını görmek zorundadır.

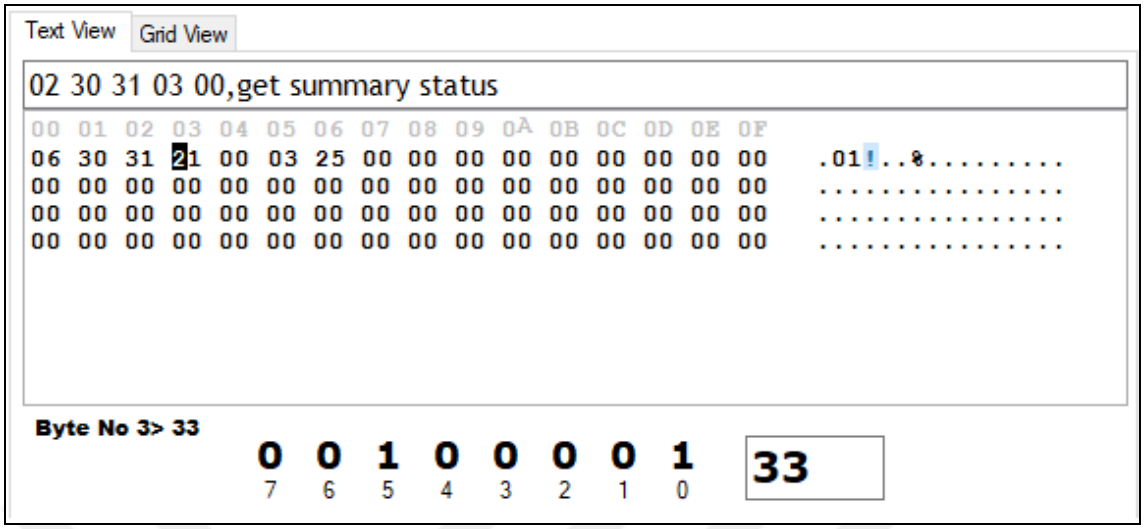
### 4.12.3 DeviceTester Kullanılarak Komut Paketinin Gönderilmesi ve Gelen Cevabın Görüntülenmesi

Oluşturulmuş komut paketini cihaza göndermek üzere Şekil 4.32’de görülen DeviceTester programında gönderilecek komut paketi seçili iken “SEND TO DEVICE” butonuna basılır.




**Şekil 4.32** DeviceTester üzerinden komut paketinin gönderilmesi

Cihazdan gelen cevap DeviceTester üzerinde Şekil 4.33'teki gibi gösterilir.




**Şekil 4.33** DeviceTester üzerinde gelen cevabın görünümü

Şekil 4.33'te görüldüğü gibi durum bilgisini gösteren byte değerinin bit görünümü için, sadece üzerine tıklamak yeterlidir.

DeviceTester kullanıldığında bir değerin bit karşılığını bulmak üzere hesap makinası, ASCII karşılığını bulmak üzere ASCII-HEX tablosu gibi araçlara ihtiyaç kalmamaktadır.

Gelen cevabı bit seviyesinde doğrudan görmenin yanında, bit değerlerini üzerine çift tıklayıp değiştirerek desimal karşılığını görmek mümkündür. Bu şekilde örneğin şu an 0 olan 2 numaralı bit değeri 1 olsaydı, hangi desimal veya HEX değerin görülmesi gerekirdi sorusuna anında cevap bulmak mümkündür. Bu kısım aynı zamanda bir "bit dizisinden desimale" dönüştürücüdür.

Sonuç olarak kullanıcı Şekil 4.30'daki tabloyu bit değerleri üzerinden yorumlama işlemini DeviceTester üzerinde daha hızlı ve hata ihtimali daha düşük bir şekilde yapabilir.

# 5. SONUÇLAR VE DEĞERLENDİRME

Gerçekleştirilen bu çalışmada TCP, UDP, SNMP, VISA, seri haberleşme gibi farklı haberleşme protokollerini destekleyen ve bu sayede farklı tipte cihazlarla haberleşebilen, onları test amaçlı kontrol ve monitör edebilen bir yazılım geliştirilmiştir.

Cihazlarla haberleşmek için yaygın olarak kullanılan programlardaki eksiklikler böyle bir çalışmanın gerekliliğini ortaya çıkarmıştır. Bu eksiklikler temel olarak üzerinde çalışılan cihazlar için komutların açıklamalı olarak kaydedilmesi, komut paketlerinin protokollere uygun bir şekilde oluşturulması, bu oluşturma esnasında protokole bağlı olarak gerekli olan checksum hesaplarının yapılması, cihazlardan gelen cevapların aynı anda ASCII, HEX, desimal formatlarda görüntülenmesi ve tarih, giden komut, gelen cevap şeklinde haberleşmenin her cihaz için ayrı dosyada loglanması gibi özellikler yaygın kullanılan programlarda bulunmamakta veya yetersiz seviyede bulunmaktadır.

Yazılım Microsoft C#.NET dili kullanılarak Visual Studio Community Edition üzerinde geliştirilmiştir. Haberleşme ve komut paketleri üzerinde işlem yapmak için alternatif olarak Python dilinin 3.6 versiyonu kullanılmıştır.

Cihaz kontrol ve monitör işlemi donanımsal ve yazılımsal çok sayıda parametreden oluştuğundan ve özellikle birden fazla sayıda cihazla çalışmak gerektiğinde, bu parametreler fazlalaşarak işin karmaşıklık miktarını arttırdığından cihazlar için kontrol yazılımı geliştirme süreci hataya, zaman kaybına ve projelerin uzamasına çok yatkındır.

Bu tezde geliştirilen program cihazlar üzerinde test edilen komutların kaydedilmesi, oluşturulması, görüntülenmesi, incelenmesi ve loglanması gibi aşamaları kullanıcıya hız ve geniş görüş açısı kazandıracak ve karmaşık bir yapısı olan cihazlarla çalışma sürecini adımları netleştirip, ihtiyaç duyulan temel yazılımsal araçları tek bir programda bir araya getirerek basitleştirecek bir şekilde tasarlanmıştır.

Buna göre program her cihaz için komutları saklamak üzere bir XML dosyası oluşturur. Bu dosya içerisinde komutlar açıklamaları ile bulunur. Komutlar programa yüklendikten sonra, üzerlerinde ekleme, düzenleme, checksum hesaplama işlemleri yapılabilir. Bu işlemler yapıldığında komutlar otomatik kaydedilir. Program gömülü olarak on iki adet checksum formatı içermektedir. Eğer bunlardan farklı bir checksum

ile çalışılacak ise, bu checksum hesaplaması Python üzerinde kod yazarak programa entegre edilebilir. Veriyi hesap makinası, tablo ve dönüştürücüye ihtiyaç bırakmayacak şekilde aynı anda ASCII, HEX, desimal formatta görmek mümkündür.

Listede mevcut komutlar kullanılan sisteme göre TCP, UDP, SNMP, seri port haberleşme yöntemlerinden biri kullanılarak cihaza gönderilebilir. Kullanıcıya komutu cihaza gönderme ile ilgili esneklik sağlamak için programa Python üzerinden haberleşme özelliği eklenmiştir ve kullanıcı Python üzerinden TCP, SNMP, VISA ve seri haberleşme yollarından biriyle cihaza komut gönderebilir.

Ayrıca program için basit bir Python kod editörü geliştirilmiştir. Bu editör ihtiyaç duyulduğunda program üzerinden açılabilmekte ve mevcut Python kodları üzerinde değişikliklik yapılabilmektedir. Bu sayede gerek checksum hesaplamaları ile ilgili kodlar, gerek haberleşme ile ilgili kodlar, bu çalışma için özel olarak geliştirilmiş bir Python editörü üzerinden güncellenebilir.

Bu çalışmada ana amaç, bu tip çalışmalarda zaman ve görüş kaybına neden olan bütün karmaşıklık ve sebepleri ortadan kaldırmak ve kullanıcının ana konuya, cihazın komut kümesini çözümleme, komutların istenen şekilde çalışması, cevapların hızlı bir şekilde analiz edilmesi ve anlaşılması gibi temel meselelere odaklanmasıdır. Bu şekilde diğer bazı açılardan da kazanımlar olacaktır. Cihaz kontrol yazılımı geliştirme sürecine giren yeni programcılar daha hızlı bir şekilde bu konuda gerekli temel seviyeye gelecekler, cihaz üzerinde yapılan çalışmalar dosyalar şekilde bir bilgi birikimi oluşturacak ve bu sayede takım halinde çalışan programcılar yaptıkları çalışmaları paylaşabileceklerdir.

Program bilgisayar dünyasında kullanılan temel formatları aynı anda gösteren ve birbirine dönüştüren özellikleri ve çok sayıda farklı haberleşme protokolünü Python açık kaynak kodları ile tek bir ortamda sunduğundan, aynı zamanda öğrenciye hızlı mesafe aldırabilecek bir eğitimsel araç olup ilgili derslerde kullanılabilir.

Bu çalışmanın geldiği son nokta, yeni geliştirme fikirleri ortaya çıkarmıştır. Komutların kaydedildiği ve görüntülendiği kısım için özel bir editör geliştirilebilir. Şu an komut ve açıklaması olan bu kısımda, komutlar üzerinde bir programlama dili ortamındaki gibi değişiklik yapılabilen, editör içinde komutları ağaç yapısı şeklinde gruplandırabilen, komutlar için detaylı açıklamalar yazılabilen, açıklama ve komutları

ayrı renklerde göstererek görsel açıdan kullanıcıyı rahatlatan bir bileşen yazılabilir.

Programda şu an zamanlayıcı kullanılarak bir cihaza sabit zaman aralıklarında komutlar göndermek mümkündür. Bu kısım için, değişken zaman aralıklarında, cihazdan gelen cevaplara ve ihtiyaca göre tanımlanan koşullara bağlı olarak otomatik komut gönderen bir mekanizma geliştirilebilir.

Programda kayıtlı olan komutların bir grafik arayüzdeki butonlara bağlandığı ve butonlar tıklandıktan sonra gelen cevapların farklı tipteki grafik göstergelerde görüntülendiği bir yapı programa eklenebilir.

Programda kayıtlı olan komut dosyaları, bu iş için geliştirilen bir web sitesi üzerinde toplanabilir. Programa "komut dosyasını siteye gönder", "sitede komut dosyası ara", "siteden komut dosyası yükle" gibi özellikler eklenebilir. Böyle bir çalışma yapılırsa, cihazları geliştiren firmalar, bu siteye komut dosyalarını yüklerler ve kullanıcılar buradan komut dosyalarını indirerek, komut yapısını çözümlemek için zaman harcamadan doğrudan cihazla haberleşmeye başlayabilirler.

# KAYNAKLAR


[1] China National Knowledge Infrastracture web site, Interface Based on PIC18F4580, 2017, (online), Available: http://en.cnki.com.cn/Article_en/CJFDTOTAL-CZGB200805014.htm

[2] Parab, J., Shelake, V., Kamat, R, Naik, G., Exploring C For Microcontrollers, Hyperterminal-Based Control (69-83), *Springer*, 2007.

[3] Abdul Aziz, N., Azura Othman K, Seroja Sarnin, S., Mohd Ali Y.I., Wireless System for Temperature Monitoring in Oil Palm Bio-laboratory, *Fifth International Conference on MEMS, NANO, and Smart Systems (ICMENS)*, Dubai United Arab Emirates 28-30 Dec.2009

[4] China National Knowledge Infrastracture web site, Based on ARM7-μCLinux and use hyperterminal to monitor two-way voltage parameters, (2017), (online), Available: http://en.cnki.com.cn/Article_en/CJFDTotal-GWDZ201011039.htm

[5] Chen, K., Ma, Y., Implementation of Remote Monitoring System Based on SNMP Protocol and Virtual Instrument for Shipborned Equipments in TT&C, Advanced Materials Research, Vol. 566, pp. 498-504, 2012

[6] Katherine, J., Network management in the world of standards: The role of the snmp protocol in managing networks, International Journal of Network Management, Vol.1, Issue 1, p.5-13, September 1991,

[7] Code Project web site, Hasan T., How To Calculate CRC in C#, 2009. (2017), (online), Available: https://www.codeproject.com/Articles/35134/How-to-calculate-a-CRC-in-Csharp

[8] CRC Calculator, (2017), (online), Available: https://www.lammertbies.nl/comm/info/crc-calculation.html

[9] Microsoft web site, (2017), (online), https://msdn.microsoft.com/en-us/library/bb311038.aspx

[10] Microsoft web site, (2017), (online), https://msdn.microsoft.com/en-us/library/bb384066.aspx

[11] Chen, S. H., Development of remote laboratory experimentation through Internet. *Proceedings of the 1999 IEEE Hong Kong Symposium on Robotics and Control*. Vol. 2. Hong Kong, 1999.

[12] Marques, R., Rocha, J., Rafael, S., Design and implementation of a reconfigurable remote laboratory, using oscilloscope/PLC network for WWW access. *IEEE Transactions on Industrial Electronics* 55.6 (2008): 2425-2432.

[13] National Instruments web site, (2017), (online), Available: http://www.ni.com/white-paper/4803/en/

[14] National Instruments web site, (2017), (online), Available: http://www.ni.com/tutorial/3271/en/

[15] MathWorks website, Using SCPI Commands from MatLab, (2017), (online), Available: https://www.mathworks.com/products/instrument/supported/scpi.html

[16] Blum, R., C# Network Programming, *Transcend Technique*, 2003.

[17] Makofske, D.B., Donahoo, M.J., Calvert, K.L., TCP/IP Sockets in C#, Practical Guide for Programmers, *Elsevier*, 2004.

[18] McMillan, M., Data Structures and Algorithms Using C#, Cambridge University Press, 2007.

[19] Weisfeld, M. The Object-Oriented Thought Process, Developer's Library, 2013.

[20] Microsoft website, (2017), (online), Available: https://technet.microsoft.com/en-us/library/bb457166.aspx

[21] HW-Group website, (2017), (online), Available: http://www.hwgroup.com/products/hercules/index_en.html

[22] Putty official website, online, (2017), Available: http://www.putty.org

[23] Packet Sender official web site, (2018), https://packetsender.com/

[24] IVI Foundation website, (2018), (online), Available: http://www.ivifoundation.org/

[25] IVI Foundation website, (2018), (online), Available: http://www.ivifoundation.org/shared_components/Default.aspx

[26] National Instruments web site ( 2018), (online) http://www.ni.com/download/ni-visa-17.0/6646/en/

[27] Xicom Technology, Serial Command Set For High Power Amplifier, Xicom Technology, 2010

# EKLER

## EK1: YAYIN LİSTESİ

Bu tezden yayımlanmış bildiri listesi aşağıda verilmiştir.

- Karadeniz, Y., Küçük, H., Development of a Novel Monitoring and Control Software for Electronic Device Testing and Analysis Purposes, IATS, 8th International Advanced Technologies Symposium, 19-21 October, 2017, Elazığ, Turkey.

# ÖZGEÇMİŞ

**Adı Soyadı** : Yasir Karadeniz

**Doğum Yeri ve Tarihi** : Düzce, 1977

**Yabancı Dili** : İngilizce, Orta Seviye Rusça, Temel Seviye Arapça

**E-Posta** : yasir@unizayn.com

**Öğrenim Durumu**

| Derece | Bölüm/Program | Üniversite/Lise | Mezuniyet Yılı |
|---|---|---|---|
| **Lise** | Elektronik Bölümü | Samsun Atakum Anadolu Teknik Lisesi | 1995 |
| **Üniversite** | Elektronik-Haberleşme Öğretmenliği | Marmara Üniversitesi | 2002 |

## İş Tecrübesi

2002 yılında araç takip sistemleri üreten bir firmada yazılımcı olarak çalışmaya başladım.

2003-2004 yılları arasında Hakkari'nin Çukurca ilçesinde Jandarma Komando Asteğmen olarak askerliğimi bitirdim.

2004 yılında televizyon kanalları ve film yapımcılarına ekipman temin eden ve televizyon stüdyoları kuran bir şirkette çalışmaya başladım. Burada iş süreçlerini otomatize eden ana muhasebe sistemine entegre bir dizi iş yönetim yazılımı ve 2018 yılı itibari ile halen kullanılmakta olan özel bir CRM yazılımı geliştirdim.

2009 yılında televizyon kanallarının uplink kısmı için sistem entegrasyonu yapan ve ürün geliştiren bir firmada çalışmaya başladım. Bu firmada yurtiçi ve yurtdışında çok sayıda televizyon kanalı için uplink sistem kontrol yazılımı ve firmanın üretmekte olduğu cihazlar için web tabanlı ve mobil kontrol yazılımları geliştirdim. Ayrıca iki adet TÜBİTAK destekli proje için sistem ve anten kontrol yazılımı geliştirdim.

2017 yılından itibaren kendi yazılım firmamda yazılım projelerine devam etmekteyim.