ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

OE-KVK
2023.

Kohán Gergely
T009287/FI12904/K

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Kohán Gergely
Szakdolgozat száma: SZD2309040923401669[redacted]
Törzskönyvi száma: T009287/FI12904/K **Neptun kódja:** [redacted]

Szak: villamosmérnöki (magyar nyelven)
Specializáció: Műszer-automatika specializáció

A dolgozat címe: Automatikusan generált kezelő felület LIN eszközökhöz
A dolgozat címe angolul: Automatically generated control panel for LIN devices
A feladat részletezése: Python alapú univerzális kezelőfelület LDF fájl alapján, ami képes LIN-en vezérlő jeleket küldeni, illetve megjeleníteni az érkező üzeneteket kvaser hardware segítségével

Intézményi konzulens neve: Sándor Tamás

A kiadott téma elévülési határideje:2025. 12. 31.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2023. 10. 24.

P.H.

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikai és Kommunikációs Rendszerek Intézet

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

.....
belső konzulens

.....
külső konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 2023.12.08

.....
hallgató aláírása

Tartalomjegyzék

1 Bevezetés

A szakdolgozatom témájaként egy Python programozási nyelven készített kezelőfelületet választottam, amely dinamikusan generálódik a bementeként kiválasztott LDF fájl alapján.

Az ötlet alapját egy LIN master node szimulálása képezi, mivel a cégnél jellemzően slave node-ként funkcionáló termékeket fejlesztenek, a készített program ezen termékek helyes működésének tesztelésében nyújt segítséget.

Munkám ötlete Metzner Miklós Illéstől (Lead Software Engineer) származik, aki munkámat végig követte és felügyelte a fejlesztés folyamatát.

A program eredetileg egy nagyobb szoftver részeként készült, amellyel képesek lennénk letesztelni minden általunk készített szoftvert. A dolgozat kizárólag ezen modul fejlesztésének menetére koncentrál.

Az első iterációban az eszköz csak a küldés és fogdás funkcionalitását tartalmazta, későbbiekben ezt kiegészítettem egy grafikus megjelenítési lehetőséggel.

2 Mozaikszavak

API - Application Programming Interface

CAN – Control Area Network

CPU – Control Processing Unit

ECU – Electronic Control Unit

GUI – Graphical User interface

LIN - Local Interconnect Network

LDF - LIN Description File

NAD – Node Address for Diagnostics

PySimpleGUI – Python Graphical User interface

MVC – (Model – View - Controller)

3 Specifikáció

3.1 Feladat részletezése

Egy "Mester" eszköz szimulálása LIN hálózaton az alábbi lépésekkel valósítható meg:

1. **LIN protokoll megértése:**
   Első lépés a LIN protokoll működésének megértése, és azon jellemzőket melyek a LIN hálózatokban használatosak.
   A LIN egy soros kommunikációs protokoll, amelyet autóipari környezetben előszeretettel alkalmaznak.

2. **Szükséges Hardver:**
   A szimuláció futtatásához szükség van egy olyan hardverre, amely támogatja a LIN kommunikációt. Ezen eszköznek képesnek kell lennie az emulációra és a LIN keretek küldésére és fogadására.

3. **LIN Kommunikáció Szoftver:**
   Válasszunk vagy fejlesszünk ki egy olyan szoftvert, amely képes emulálni a Mester eszközt. Ez a szoftver teszi lehetővé a LIN keretek küldését és fogadását, valamint a kommunikáció ellenőrzését.

4. **LIN Konfiguráció:**
   LIN hálózat konfigurációja és a szükséges paraméterek beállítása. A konfiguráció magában foglalja a sebességet, az eszközök címzését, az adatátviteli sebesség beállítását és az esetleges ellenőrző összegek használatát.

5. **Szimuláció futtatása:**
   A szimulációt elindítva az emulált Mester eszköz képes lesz LIN kereteket küldeni a LIN hálózatra és fogadni a válaszokat. Ezenkívül a szimuláció során a Mester eszköz viselkedését is megfelelően be kell programozni a célrendszer működésének szimulálására.

6. **Hibakezelés:**
   A LIN hálózatokban gyakran szükség van hibakezelésre és az esetleges hibák ellenőrzésére. A szimulált Mester eszköznek képesnek kell lennie a hibás keretek felismerésére és kezelésére.

7. **Tesztelés:**
   A szimulált Mester eszköz működését tesztelni kell a valós környezetben vagy más LIN eszközökkel való kommunikáció során. Ellenőrizni kell, hogy a szimulált Mester eszköz helyesen viselkedik-e és megfelel-e az elvárásoknak.

Ezen lépések megvalósításához elengedhetetlen a LIN hálózatok mély és átfogó ismerete, a felhasznált hardverek és szoftverek alapos ismerete. A megfelelő szimuláció lehetővé teszi a fejlesztők számára, hogy a LIN hálózatokon működő rendszereiket teszteljék anélkül, hogy a valós környezetben működtetnék az eszközöket.

3.2 Eszközök részletezése

3.2.1 neoVI RED

1. ábra NeoVI készülék [1]

A neoVI RED egy olyan fejlett eszköz, amely kifejezetten tervezve lett a járműelektronikai rendszerek mélyebb vizsgálatára és analizálására. Ezt a berendezést mérnökök és fejlesztők használják annak érdekében, hogy alapos ismereteket szerezzenek a járművek elektronikai rendszereinek működéséről és kommunikációjáról.

Az eszköz képes egyszerűen csatlakozni a jármű CAN (Controller Area Network) hálózatához, amely kulcsfontosságú a modern járművek elektronikai rendszereiben. A CAN hálózat lehetővé teszi az elektronikai vezérlőegységek közötti hatékony adatcserét és kommunikációt. A neoVI RED képes megjeleníteni és rögzíteni ezen hálózatok valós idejű tevékenységét, lehetőséget adva a felhasználóknak a részletes elemzésre és hibakeresésre.

A funkcionális gazdagsága mellett a neoVI RED rendelkezik továbbá kiterjedt protokoll-támogatással. Ez azt jelenti, hogy nemcsak a CAN hálózattal, hanem más, a járművekben alkalmazott kommunikációs protokollokkal is képes megbirkózni, biztosítva ezzel a széleskörű alkalmazhatóságot.

Az eszköz egyedi előnye, hogy lehetőséget kínál a járműelektronikai rendszerek alapos felügyeletére és elemzésére, ami különösen értékes a fejlesztési és tesztelési folyamatok során. A neoVI RED segítségével a mérnökök és szakemberek hatékonyan diagnosztizálhatnak problémákat, optimalizálhatják a rendszereket, és növelhetik a járműelektronikai fejlesztések hatékonyságát.

3.2.2 Baby-LIN-II

2. ábra Baby-LIN eszköz [2]

A Baby-LIN-II a LIN protokollt alkalmazza a járműipari alkalmazásokban. A LIN protokoll egy olyan rendszer, melyet kifejezetten az autóiparban használnak, és lehetővé teszi a járműelektronikai rendszerek közötti adatátvitelt és kommunikációt.

Ez a speciális eszköz kulcsfontosságú a járműfejlesztési és tesztelési folyamatokban. A LIN protokoll segítségével kommunikál az egyes járműelektronikai vezérlőegységekkel, lehetővé téve a fejlesztők és mérnökök számára, hogy megjelenítsék és vezéreljék azokat.

Az eszköz számos funkciót kínál, beleértve a LIN hálózatok diagnosztikáját, a kommunikációs sávszélesség monitorozását és az eseménynaplózást. Emellett segít a hibakeresésben és a járműelektronikai rendszerek optimalizálásában a fejlesztési ciklus különböző szakaszaiban.

Az alkalmazása kiterjedhet a prototípusoktól a sorozatgyártásig, hozzájárulva a járműelektronika megbízhatóságához és hatékonyságához. Ezáltal az eszköz elősegítheti a modern járművek fejlesztését és gyártását, segítve a mérnököket és fejlesztőket a járműelektronikai rendszerek optimalizálásában és tesztelésében.

3.2.3 VN1630A

3. ábra 2 és 4 csatornás VN1630A eszköz [3]

4. ábra 2 csatornás VN1630 Log eszköz [3]

Az említett eszköz kiemelkedő szerepet tölt be a járműelektronikai fejlesztési folyamatokban. Szorosan integrálódik a járművek hálózati kommunikációs rendszerébe, lehetővé téve a mérnökök és fejlesztők részletes vizsgálatát és tesztelését.

Ez a berendezés kifejezetten a Controller Area Network és a Local Interconnect Network protokollok támogatására lett tervezve, amelyek kulcsfontosságúak a járműelektronikai rendszerek közötti adatátvitelhez és kommunikációhoz. A felhasználóknak lehetőségük van valós idejű adatok gyűjtésére, hálózatok figyelésére és részletes elemzésekre.

Az eszköz nemcsak segíti a járműelektronikai rendszerek vizsgálatát, hanem kiválóan alkalmas hibakeresésre és prototípusok tesztelésére is. Hozzájárul a járműelektronika teljesítményének és megbízhatóságának optimalizálásához a fejlesztési ciklus különböző szakaszaiban.

A rugalmas kialakításnak és a széleskörű protokoll-támogatásnak köszönhetően könnyen integrálható különféle fejlesztői környezetekben. Mindezek együttesen elősegítik a járműipari szakemberek hatékonyabb járműelektronikai fejlesztését és tesztelését.

3.2.4 Kvaser hardver

5. ábra kvaser eszköz [4]

A Kvaser Hybrid 2xCAN/LIN eszköz egy rendkívül hasznos eszköz a járműelektronikai fejlesztési és tesztelési területeken. Ez az eszköz két alapvető protokoll, a Controller Area Network és a Local Interconnect Network támogatásával rendelkezik, lehetővé téve a mérnökök és fejlesztők számára, hogy széleskörűen vizsgálják a járművek kommunikációs hálózatait.

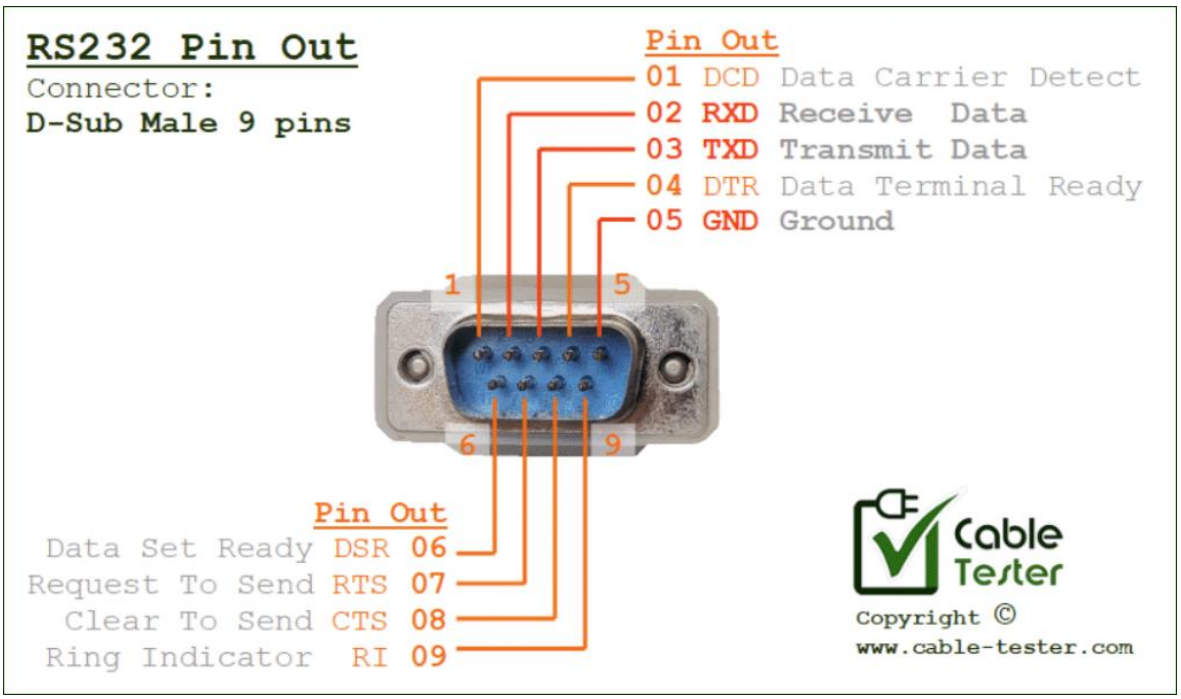
A Kvaser Hybrid 2xCAN/LIN eszköz két különálló hálózati interfésszel rendelkezik, amelyek lehetővé teszik a CAN és a LIN hálózatok egyidejű figyelését és kommunikációját. A CAN protokoll széles körben használt a járműelektronikában, míg a LIN protokoll kisebb adatsebességgel rendelkezik, de ideális az olyan alacsonyabb szintű eszközök, mint például az érzékelők és a kapcsolók számára.

Ez az eszköz megkönnyíti a valós idejű adatgyűjtést és az események rögzítését mindkét hálózaton. A hibakeresés és a rendszerfejlesztés során a Kvaser Hybrid 2xCAN/LIN segítséget nyújt a részletes elemzésekben és a hálózati kommunikációs folyamatok megértésében.

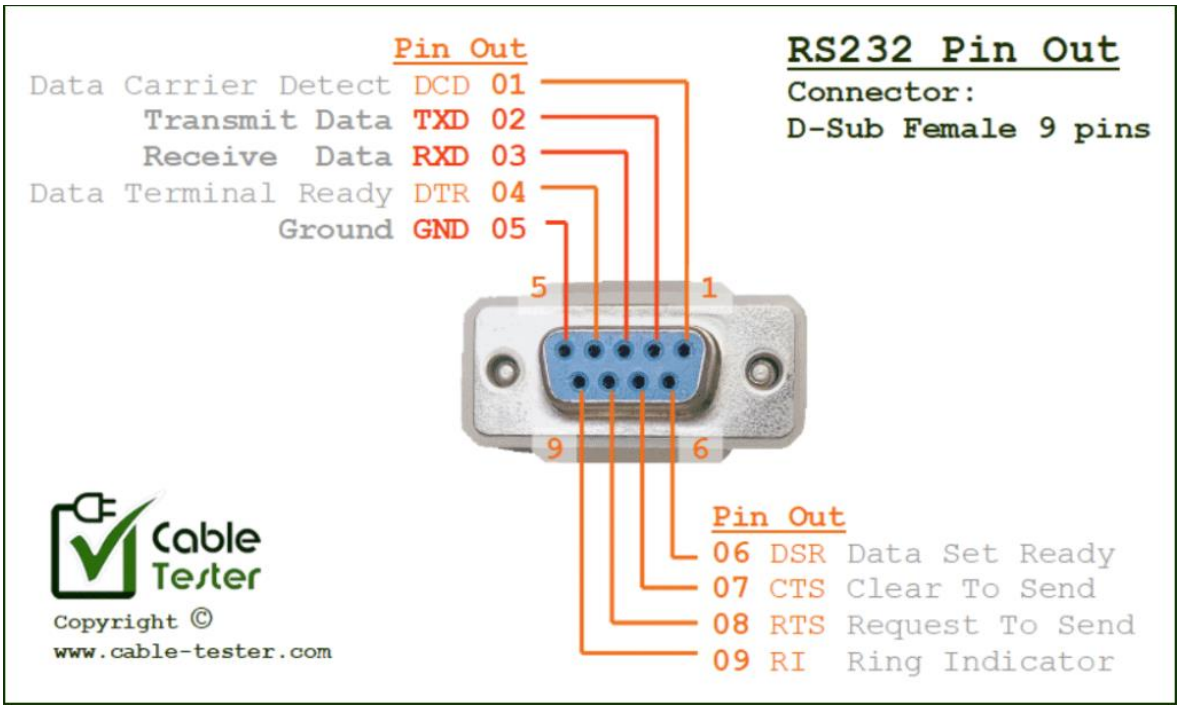
A rugalmas kialakításának köszönhetően az eszköz alkalmazkodik a fejlesztői környezethez, és lehetővé teszi az integrációt különféle tesztrendszerekbe és alkalmazásokba. Összességében a Kvaser Hybrid 2xCAN/LIN eszköz kiemelkedő segítséget nyújt a járműelektronikai területen dolgozó szakemberek számára, akiknek szükségük van a hálózati kommunikációs folyamatok mélyebb megértésére és fejlesztésére.

Ezen tulajdonságok teszik nélkülözhetetlenné a Kvaser Hybrid 2xCAN/LIN-t minden autóipari kommunikációval foglalkozó mérnök számára.

3.3 A D-SUB csatlakozóról részletesen



6. ábra D-Sub Male csatlakozó [5]



7. ábra D-Sub Female csatlakozó [5]

A D-SUB 9 csatlakozó egy olyan elterjedt és sokoldalú csatlakozó, amelyet gyakran használnak a különböző elektronikai és informatikai eszközök kapcsolásához és kommunikációjához. A " D-SUB " név egy olyan családot jelöl, amelynek tagjai a D-alakú csatlakozó formátumot használják, és az "9" a csatlakozó lábainak számát jelzi. Ez a család különböző méretű D-SUB csatlakozókat tartalmaz, a D-SUB 9 pedig egyike a legelterjedtebbeknek.

Általában számítógépek, monitorok, nyomtatók és egyéb perifériák összekapcsolására használatosak.

A D-SUB 9 csatlakozók leírását és szabványait az EIA-232 (RS-232) szabvány határozza meg. Az RS-232 az "Elektronikus Ipari Szövetség" (Electronic Industries Alliance) által kidolgozott szabvány, amely a soros kommunikációt és interfészeket szabályozza. A D-sub 9 csatlakozó gyakran használják az RS-232 interfészen keresztüli kapcsolódásokhoz, például soros portokon keresztül a számítógépek és perifériák között.

Az RS-232 szabvány rögzíti a csatlakozók megfelelő pin-kiosztását és a kommunikáció során használt elektromos jelek paramétereit. A D-SUB 9 csatlakozóknál például a különböző lábak az adatok, a földelés és az ellenállások különböző funkcióit szolgálják. Az RS-232 szabványban részletesen le vannak írva ezek a funkciók és a kommunikációhoz szükséges protokollok.

Mivel az RS-232 és a D-SUB 9 csatlakozók olyan elterjedtek és fontosok az elektronikai iparágban és az informatikában, a szabványok meghatározzák azokat a normákat és követelményeket, amelyek biztosítják a kompatibilitást és a megbízható kommunikációt a különböző eszközök között.

3.4 A LIN-ről részletesen

ISO 17987-1:2016 ebben a szabványban található minden információ, ami szükséges egy LIN kommunikációhoz. [6]

Alap állítások:

- A LIN az OSI-modell szerinti átfogó koncepció, ahol a fizikai, adatkapcsolati, hálózati és alkalmazási rétegek a specifikáció által vannak lefedve.
- • A LIN fizikai rétege az ISO 9141 (a K-vonal) alapján készült.
- • Mester / szolga szervezet
- • Egyetlen vezeték és földelés
- • Időzített ütemezés
- • 1-20 kbit/s sebesség
- • Domináns / recesszív bit-ek
- • Soros, bájt alapú kommunikáció
- • Max. 40 m vezetékhossz
- • Az LIN szervezet által meghatározott szabvány: http://www.lin-subbus.org [7]

A LIN protokoll a Volcano-Lite technológián alapul, amit a Volvo kiváltásaként létrehozott Volcano Communications Technology (VCT) fejlesztett ki. Mivel más autógyártók is kerestek egy költséghatékony alternatívát a CAN protokollhoz, létrehozták a LIN szövetséget. Ebben a szövetségben dolgozták ki az első LIN protokollt (1.0) 1999 közepén. Ezt a protokollt 2000-ben kétszer is frissítették. Az LIN 1.3 verzió 2002 novemberében jelent meg, főként a fizikai rétegen történtek változások. A legújabb verzió, az LIN 2.2A, 2010-ben látott napvilágot. Az LIN 2.0 verzióval komoly változások érkeztek, és új funkciók is bekerültek, mint például a diagnosztikai lehetőségek. Ezek a változtatások elsősorban az off-the-shelf (polcról vásárolható) Slave (kiszolgált) eszközök egyszerűbb használatát célozzák meg.

A LIN protokoll a CAN és az SAE J1850 protokollokkal kiegészítő megoldást nyújt olyan alkalmazások számára, amelyek nem időkritikusak, vagy nem igényelnek rendkívül magas hibatűrő képességet, mivel a LIN kevésbé megbízható, mint a CAN. A LIN fő célja az egyszerű használat és egy költséghatékonyabb alternatíva nyújtása a CAN-hez képest. Példák olyan területekre, ahol a LIN jelenleg használatos vagy potenciálisan alkalmazható egy autóban: ablakemelők, tükrök, ablaktörlők és ülésfűtés.

3.5 A Python-ról részletesen

A Python egy népszerű programozási nyelv, amely számos előnnyel rendelkezik, és egyre szélesebb körben használják a világ különböző területein. A Python egy általános célú nyelv, amelyet Guido van Rossum hozott létre 1989-ben. Azóta folyamatosan fejlődött és vált az egyik legfontosabb eszközzé a szoftverfejlesztők, tudományos kutatók és adatelemzők számára.

A Python könnyen tanulható és olvasható nyelv, amelynek szintaxisa intuitív és egyszerű, ezen tulajdonság hozzájárul, hogy kevesebb kódsorral és kevesebb hibalehetőséggel lehessen programokat létrehozni.

Emellett a Python rendkívül rugalmas, előre elkészített modulok és könyvtárak széleskörben rendelkezésre állnak, melyek különböző feladtok hatékony és gyors végrehajtásában nyújtanak segítséget.

A Python ismert a tudományos számításokra, a webfejlesztésre, az adatkezelésre és az automatizálásra alkalmas moduljairól, mint például a NumPy, Pandas, Django és Selenium.

A Python széles körű közössége és támogatottsága révén a fejlesztők számára könnyű segítséget és forrásokat találni.

A nyelv gyakran első választás mind a kezdő programozók és mind a tapasztalt fejlesztők körében, mivel lehetővé teszi az egyszerű alkalmazások gyors elkészítését és komplex projektek fejlesztését egyaránt.

Tökéletesen alkalmazkodik a különböző igényekhez és kihívásokhoz, és szinte bármilyen területen használható, így választásával a fejlesztők könnyen hatékony, olvasható és karbantartható kódot írhatnak. Python nyelvvel a lehetőségek szinte végtelenek, és ez az egyik oka annak, hogy olyan népszerű és elterjedt a programozók körében.

3.6 A C#-ról részletesen

C# (C-sharp) egy erőteljes és széles körben használt programozási nyelv, amelyet a Microsoft fejlesztett ki a .NET keretrendszerhez. C# egy modern és objektumorientált nyelv, amelyet gyakran használnak asztali alkalmazások, webalkalmazások és játékok fejlesztéséhez. A C# nyelvet általában "C sharp"-ként ejtik, és elterjedt az üzleti alkalmazások és az ipari rendszerek fejlesztésében.

Az egyik kiemelkedő jellemzője a C#-nak, hogy támogatja az erős típusosságot és a statikus típusellenőrzést, ami lehetővé teszi a hibák előzetes felderítését a fejlesztés során. Ez segít a kód stabilitásának és megbízhatóságának biztosításában.

Könnyen tanulható és rendelkezik egy egyszerű, olvasható szintaxissal. A fejlesztők könnyen kezelhetik a különböző adattípusokat, objektumokat és osztályokat, ami lehetővé teszi az alkalmazások könnyű tervezését és karbantarthatóságát. Emellett a C# teljeskörű fejlesztői környezettel (IDE) rendelkezik, például a Visual Studio, amely segíti a fejlesztőket a kódszerkesztésben, hibakeresésben és projektkezelésben.

Az alkalmazásokat különböző platformokon futtathatják, beleértve a Windows-t, az iOS-t, az Androidot és a Linuxot is, azáltal, hogy a .NET Core és a .NET 5/6 keretrendszerekkel kombinálják. Ez teszi lehetővé a C# használatát széles körű alkalmazások fejlesztéséhez.

Összességében a C# egy erőteljes és sokoldalú programozási nyelv, amelynek számos alkalmazása van az informatikában, a fejlesztők pedig gyakran választják az üzleti és az ipari projektekhez egyaránt. Az aktív fejlesztői közösség és a Microsoft támogatása révén a C# továbbra is népszerű és növekvő nyelv a szoftverfejlesztés területén.

3.7 A két nyelv össze hasonlítása

A Python és a C# két különböző programozási nyelv, amelyeket különböző célokra és környezetekben használnak. Itt néhány kulcsfontosságú különbség és hasonlóság a két nyelv között:

1. Cél:

Python: Python egy általános célú nyelv, amelyet sokféle alkalmazásban használnak, beleértve a webfejlesztést, tudományos számításokat, gépi tanulást, automatizálást és sok egyéb területet.

C#: C# erősen kapcsolódik a Microsoft ökoszisztémához, és gyakran használják asztali alkalmazások, játékok és üzleti alkalmazások fejlesztéséhez Windows környezetben.

2. Szintaxis:

Python: Python könnyen tanulható és olvasható nyelv, amelynek szintaxisa egyszerű és intuitív. A Python számára nem szükséges pontosvesszők használata, és az indentálás (bekezdetek használata) meghatározza a kódblokkokat.

C#: C# is egyszerű és olvasható nyelv, de erősebb típusrendszerrel rendelkezik, ami statikus típusellenőrzést tesz lehetővé. C# használatához pontosvesszőket kell használni a kódban.

3. Típusrendszer:

Python: Python egy dinamikus típusú nyelv, ami azt jelenti, hogy a változók típusát a futási időben határozza meg. A típusdeklarációk opcionálisok.

C#: C# egy erősen típusos nyelv, és a típusokat a fordítási időben ellenőrzi. A típusdeklarációk kötelezőek.

4. Platformfüggetlenség:

Python: Python alapvetően platformfüggetlen, és számos operációs rendszeren és környezetben futtatható.

C#: C#-t általában a Windows platformhoz kötik, de a .NET Core és a .NET 5/6 lehetővé teszi a C# alkalmazások futtatását más platformokon is.

5. Közösség és Környezetek:

Python: Pythonnak egy nagy és aktív közössége van, és számos külső könyvtár és modul elérhető a különböző feladatokhoz. A Python fejlesztői környezetek között a Jupyter Notebook, a Visual Studio Code és az Anaconda találhatók.

C#: C# a Microsoft támogatásával rendelkezik, és a Visual Studio a leggyakrabban használt fejlesztői környezet a C# fejlesztéshez.

Mindkét nyelvnek megvannak a saját erősségei és alkalmazási területei. A választás a konkrét projekt céljától és követelményeitől függ. Python könnyen tanulható és gyors prototípusfejlesztést tesz lehetővé, míg C# erőteljes választás lehet Windows alapú asztali alkalmazásokhoz és játékokhoz.

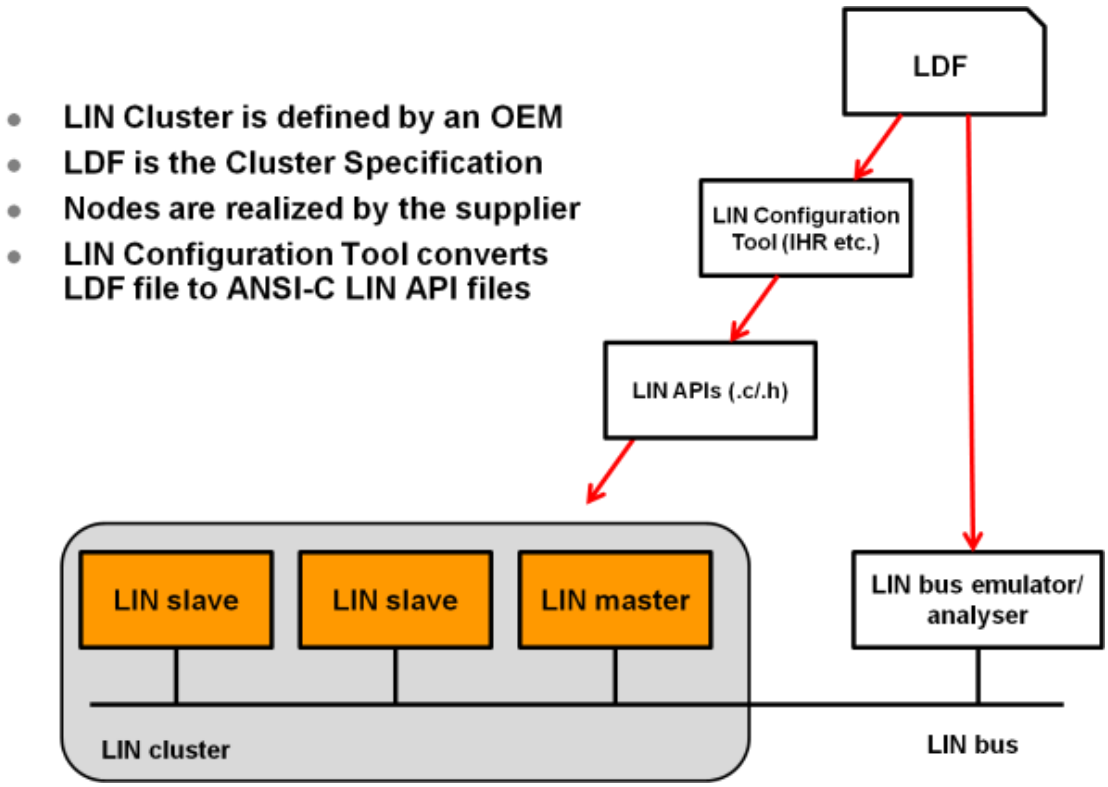
3.8 Az LDF részletesen

Az LDF egy olyan fájlformátum, amely a LIN kommunikációs protokollhoz kapcsolódik, és nem egy szabvány, hanem egy protokoll leíró fájlformátum. Az LDF fájlok használata a LIN hálózatok tervezéséhez és konfigurálásához kapcsolódik, és tartalmazza a LIN hálózathoz kapcsolódó eszközök, változók és üzenetek leírásait.

Az LDF fájlok a LIN hálózatokat használó fejlesztők és mérnökök közötti információcserét segítik elő. Az LDF fájlok leírják, hogy az adott LIN hálózat hogyan kommunikál, milyen eszközök és változók vannak jelen, és milyen üzeneteket küldenek és fogadnak az eszközök.

A fájlokat általában a LIN eszközök és alkalmazások konfigurálásához használják, és ezek az eszközök a protokoll specifikációinak megfelelően működnek. Az LDF fájlok lehetővé teszik a LIN hálózatok előzetes tervezését és tesztelését, mielőtt azokat valós környezetben használnák.

Fontos megjegyezni, hogy az LDF fájlok specifikációi és formátuma nem része egyetlen szabványnak, hanem a LIN Consortium és más kapcsolódó ipari szervezetek által kidolgozott ajánlásoknak és gyakorlatoknak felelnek meg. Az LDF fájlok tartalmazhatnak információkat az általánosan elfogadott LIN protokoll specifikációinak megvalósításához.



8. ábra Adat folyamatábra [8]

3.9 Saját LDF bemutatása

3.9.1 Fejléc

A fejléc, amely tartalmazza az általános leírást a fájlról. Mikor lett létrehozva, a mester ECU nevét a fájl nevét.

```
/**************************************************************************/
/*                                                                        */
/*  Description:   LIN Description file                                   */
/*  System:        University                                             */
/*  Cluster:       Example_file                                           */
/*  Master:        Seatheating                                            */
/*  Generated at:  2023-09-12 20:46                                       */
/*                                                                        */
/**************************************************************************/
```

3.9.2 Protokoll leírás

A LIN protokoll leírása melyik verziót használja, milyen sebességgel.

```
LIN_description_file;
LIN_protocol_version = "2.0";
LIN_language_version = "2.0";
LIN_speed = 19.2 kbps;
```

3.9.3 Csomópontok

A csatornán résztvevő „csomópontok”, avagy az ECU-k, amik kommunikálnak a buszon, megadva a periódus idejüket is.

```
Nodes {
    Master: BCM, 5.000 ms, 0.500 ms;
    Slaves: Example_ecu;
}
```

3.9.4 Jelek

A jelek, minden jel, ami részt vesz a buszon, mindegyik ECU hozzáférhet bármelyik jelhez innen. Tartalmazza a nevét hosszát, kezdő értékét és hogy ki küldi kinek.

```
Signals {
        Example_ActvReq: 3, 0, BCM, Example_ecu;
        Example_CabnTemp: 8, 254, BCM, Example_ecu;
        Example_CmdField_Req: 3, 0, BCM, Example_ecu;
        Example_ErrClr: 1, 0, BCM, Example_ecu;
        Example_Humidity: 8, 0, BCM, Example_ecu;
        Example_Ventreq: 1, 0, BCM, Example_ecu;
        Example_resperr: 1, 0, Example_ecu, BCM;
        Example_ActvReq_2: 3, 0, BCM, Example_ecu;
        Example_CabnTemp_2: 8, 254, BCM, Example_ecu;
        Example_CmdField_Req_2: 3, 0, BCM, Example_ecu;
        Example_ErrClr_2: 1, 0, BCM, Example_ecu;
        Example_Humidity_2: 8, 0, BCM, Example_ecu;
        Example_Ventreq_2: 1, 0, BCM, Example_ecu;
        Example_resperr_2: 1, 0, Example_ecu, BCM;
}
```

3.9.5 Diagnosztika

```
Diagnostic_signals {
        MasterReqB0: 8, 0;
        MasterReqB1: 8, 0;
        MasterReqB2: 8, 0;
        MasterReqB3: 8, 0;
        MasterReqB4: 8, 0;
        MasterReqB5: 8, 0;
        MasterReqB6: 8, 0;
        MasterReqB7: 8, 0;
        SlaveRespB0: 8, 0;
        SlaveRespB1: 8, 0;
        SlaveRespB2: 8, 0;
        SlaveRespB3: 8, 0;
        SlaveRespB4: 8, 0;
        SlaveRespB5: 8, 0;
        SlaveRespB6: 8, 0;
        SlaveRespB7: 8, 0;
}
```

3.9.6 Keretek

Leírja a buszon lévő keretek nevét azonosító számát, illetve, hogy ki küldi, milyen hosszú a keret és hogy milyen jeleket tartalmaz és nem utolsó sorban a jeleknek a kezdő bitjével, azaz, hogy a jel hányadik biten kezdődik az üzeneten vagy kereten belül.

```
Frames {
        Example_REQ: 0x09, BCM, 8 {
                Example_ActvReq, 1;
                Example_ErrClr, 24;
                Example_Humidity, 8;
                Example_CabnTemp, 16;
                Example_CmdField_Req, 56;
                Example_Ventreq, 0;
        }
        Example_RES: 0x10, Example_ecu, 8 {
                Example_ActvReq_2, 1;
                Example_CabnTemp_2, 16;
                Example_CmdField_Req_2, 56;
                Example_ErrClr_2, 24;
                Example_Humidity_2, 8;
                Example_Ventreq_2, 0;
        }
}
```

3.9.7 Diagnosztika keretek

Leírja, hogy hogyan néznek ki a diagnosztikai a Mester, illetve a Slave ECU keretek. Megadja milyen hosszan, milyen jelek vannak benne, illetve a jeleknek a kezdő bitjét, mint az előző részben is a normál kereteknél, viszont itt ki kell emelni, hogy minden normál csomag „single frame" csak 8 byte-ot használhat fel a protokoll szerint. Ennél hosszabb kereteknél „consecutive frame" -eket, azaz folytonos csomagokat kell használni. Minden típusú csomagnál, a fel nem használt byte-okat 0xFF -el kell kitölteni.

```
Diagnostic_frames {
        MasterReq: 60 {
                MasterReqB0, 0;
                MasterReqB1, 8;
                MasterReqB2, 16;
                MasterReqB3, 24;
                MasterReqB4, 32;
                MasterReqB5, 40;
                MasterReqB6, 48;
                MasterReqB7, 56;
        }
        SlaveResp: 61 {
                SlaveRespB0, 0;
                SlaveRespB1, 8;
                SlaveRespB2, 16;
                SlaveRespB3, 24;
                SlaveRespB4, 32;
                SlaveRespB5, 40;
                SlaveRespB6, 48;
                SlaveRespB7, 56;
        }
}
```

3.9.8 Csomópont leírás

Egy-egy csomópont, milyen protokollt használ, illetve mi a konfigurált NAD-ja, ami a diagnosztikai üzenetek azonosításához szükségesek. Tartalmazza még a termék azonosítóit és a LIN protokoll hiba esetén küldött hiba jelet.

```
Node_attributes {
        Example_ecu {
                LIN_protocol = "2.0";
                configured_NAD = 0x50;
                product_id = 0x100, 0x01, 0;
                response_error = Example_resperr;
                configurable_frames {
                        CCS_REQ = 0x28;
                        CCS_RESP = 0x29;
                }
        }

}
```

3.9.9 Ütemezési táblázatok

Leírja milyen keretek milyen időközönként jelennek meg a buszon. A diagnosztikába nem lehet a normál keretek közül egyiket sem választani ugyanis akkor az ECU kiszámíthatatlanul működne.

```
Schedule_tables {
        AllModulesFitted {
        Example_REQ delay 30.000 ms;
        Example_RES delay 10.000 ms;
        }
        Just_diag_exmp {
                MasterReq delay 15.000 ms;
                SlaveResp delay 15.000 ms;
        }
}
```

3.9.10 Jelek kódolása

A jelek kódolási típusai, fizikai értékek, ami a buszon nem ilyen formában jelenik meg mert ott „nyers” (raw) üzenetek szerepelnek. Ellenben, ha felhasználó szemmel szeretnénk értelmezni a minimum és maximum értékeket kell figyelembe vennünk ugyanis ezekből a tartományokból lehet kiszámolni az értékeket.

Ezt a számot, amit a LIN-ről olvasunk, meg kell szorozni egy faktorral és hozzá kell adni egy ofszetet,
így a végén kaphatunk például egy számot, ami Celsius fokban értendő fizikálisan, mindazonáltal a végeredmény mindenki számára egyértelmű.

Logikai értékeket is megadhatunk itt megszabhatunk bármit, diszkrét értékekkel például a 252 egy magas hőmérséklet érzékelés amikor ugyanis a tartományon kívülre esik a mérés.

Vezérlő jeleknél célszerű csak logikai értékeket megadni hiszen ezek egyszerű 1 byte-os értékek.

```
Signal_encoding_types {
        Example_CabnTemp_enc {
                physical_value, 0, 250, 0.5, -40.0, "DegC";
                logical_value, 252, "Over Temperature Detected";
                logical_value, 253, "Not Installed";
                logical_value, 254, "Standby";
                logical_value, 255, "SNA";
        }
        Example_commad_enc {
                logical_value, 0, "Standard Response";
                logical_value, 1, "Diagnostic Report";
                logical_value, 2, "Hardware Part Number";
                logical_value, 3, "Software Part Number";
                logical_value, 4, "Serial Number";
        }
        Example_Ventreq_enc {
                logical_value, 0, "OFF";
                logical_value, 1, "Low_Level";
                logical_value, 2, "Medium_Level";
                logical_value, 3, "High_Level";
                logical_value, 4, "Extra_High_Level";
```

```
        }
            Example_ActvReq_enc {
            logical_value, 0, "OFF";
            logical_value, 1, "Low_Level";
            logical_value, 2, "Medium_Level";
            logical_value, 3, "High_Level";
            logical_value, 4, "Extra_High_Level";
        }
        Example_ErrClr_enc {
            logical_value, 0, "Do_Nothing";
            logical_value, 1, "Clear_error";

        }
        Example_Humidity_enc {
            physical_value, 0, 250, 0.5, -40.0, "DegC";
            logical_value, 252, "Over Temperature Detected";
            logical_value, 253, "Not Installed";
            logical_value, 254, "Standby";
            logical_value, 255, "SNA";
        }
}
```

3.9.11 Jelábrázolás

A jelábrázolást itt kell megadni, hogy melyik jelhez milyen dekódolás típus tartozik. Egy típushoz akár több jelet is lehet illeszteni, ha azonos az értékek, például egy jobb oldali vagy bal oldali fűtés eszköz bekapcsolása elképzelhető, hogy ugyanazon a típuson lesz, de lehet külön-külön is létrehozni ezeket.

```
Signal_representation {
        Example_commad_enc : Example_CmdField_Req;
        Example_CabnTemp_enc: Example_CabnTemp;
        Example_Ventreq_enc :Example_Ventreq;
        Example_ActvReq_enc : Example_ActvReq;
        Example_ErrClr_enc : Example_ErrClr;
        Example_Humidity_enc : Example_Humidity;
}
```

3.10 Összegzés

Miért pont a Kvaser hardver-t választottam ?

A cégnél több féle eszköz is használt a projektek megvalósítása során mint például, Babylin, neoVI, Vector VN1630A,VN1640A,Kvaser ezeknek összehasonlítása az alábbi táblazatba került :

Eszköz	Ár	Automatizálás	Programnyelv	Rendelkezésre állás	Ismeretség cégen belül	Összegzés
VN1630A	-	+	0	+	+	2
Baby-LIN-II	+	-	-	0	0	-1
neoVI	-	+	-	-	-	-3
Kvaser	+	+	+	+	+	4

1. táblázat Eszközök összehasonlítása

A Kvaser többféle programnyelvet támogat, melyek közül a Python-t választottam a következő szempontok alapján:

Nyelv	Ár	Ismertség cégen belül	Hatékonyság	Támogatás	Összegzés
Python	+	+	0	+	3
C#	-	-	+	+	0

2. táblázat Programnyelvek összehasonlítása

A különböző eszközök tulajdonságait megvizsgálva döntöttem a Kvaser hardware használata konfigurálható saját igényekre.

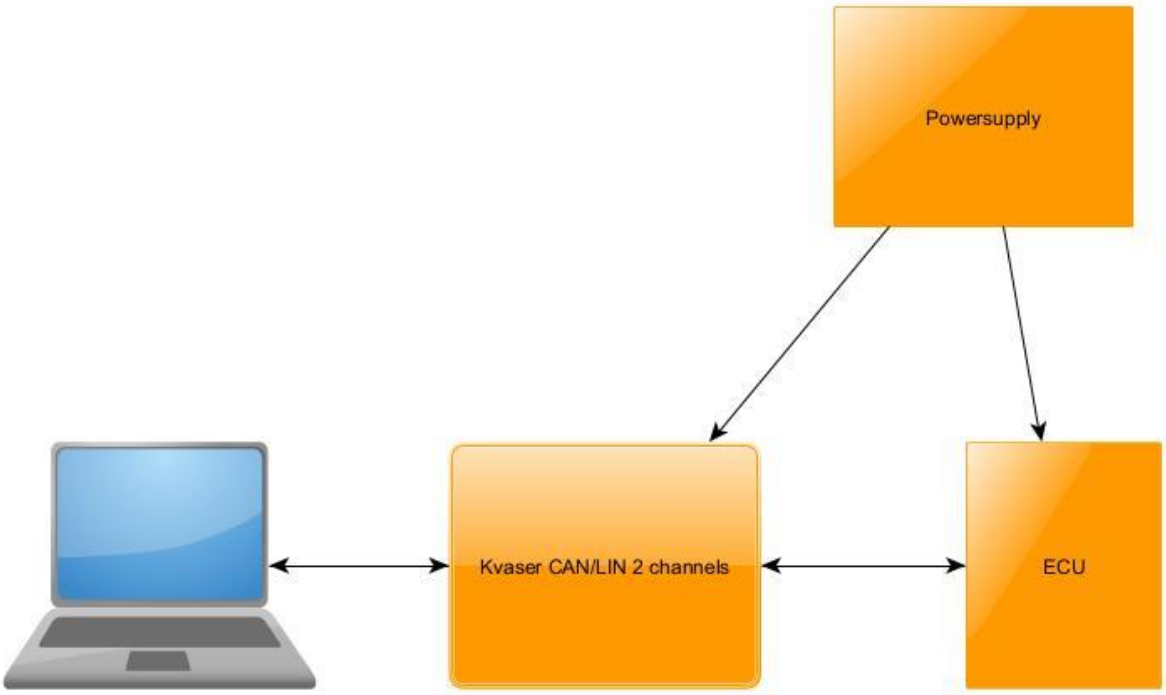
Támogatja a C#-ot és a Pythont is.

A babylin bár olcsó megoldás, de csak limitáltan konfigurálható saját programmal, másrészt nem automatizálható, manuálisan lehet csak konfigurálni.

A Vector és a neoVI hardverek használatának legfőbb hátránya hogy drágák a többi lehetőséghez képest.

mellett. Mert egy könnyü egyszerű szerkezet, bárki tudja használni illetve nagyon könnyen Logikai rendszerterv

3.11 Fizikai rendszerterv



9. ábra Logikai rendszerterv (Saját ábra)

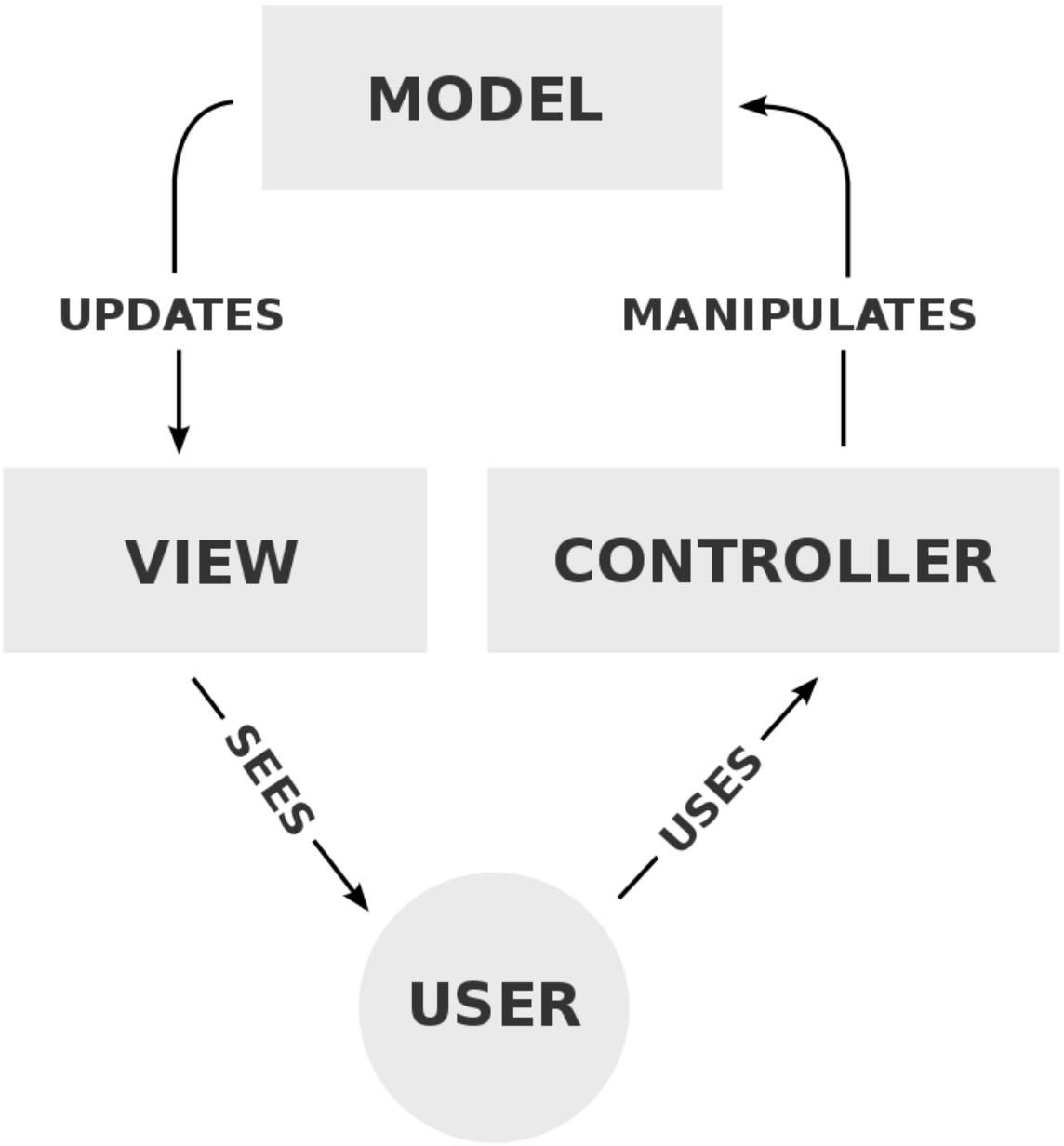
A rendszer működéshez négy eszközre van szükség, az ábrán látható elrendezés alapján, a tervezési fázisban fontos szempont volt, hogy a szükséges technikai eszközök könnyen elérhetőek és helytakarékosak legyenek, így segítve a széleskörű felhasználhatóságot.

A rendszer egyszerűsége továbbá támogatja a napjainkban jellemző hibrid vagy távoli munka végezést is, lehetőség van bárhonnan tesztelni az adott eszközt.

További jelentős feltétel volt, hogy a felhasznált driverek széleskörben támogassák az előforduló számítógép konfigurációkat, mind a driver telepítés mind az elkészített alkalmazás futtatása szempontjából.

Fontos megjegyezni, hogy a feszültségforrást nem elegendő szimplán a tesztelt eszköznek biztosítani, hanem a Kvaser hardverére is muszáj feszültséget kötni, ugyanis ennek elmulasztása során a meghajtón nem lesz feszültség, amely nélkül az nemképes fogadni és küldeni a jeleket az adott buszon, erre az esetre a szoftverben implementálva lett egy hibajelzés, ha enélkül indítanánk az ütemezőt az nem fog megfelelően működni, ilyenkor célszerű elsőként ellenőrizni a megfelelő feszültség szinteket a használt konfigurációban.

3.12 Szoftver



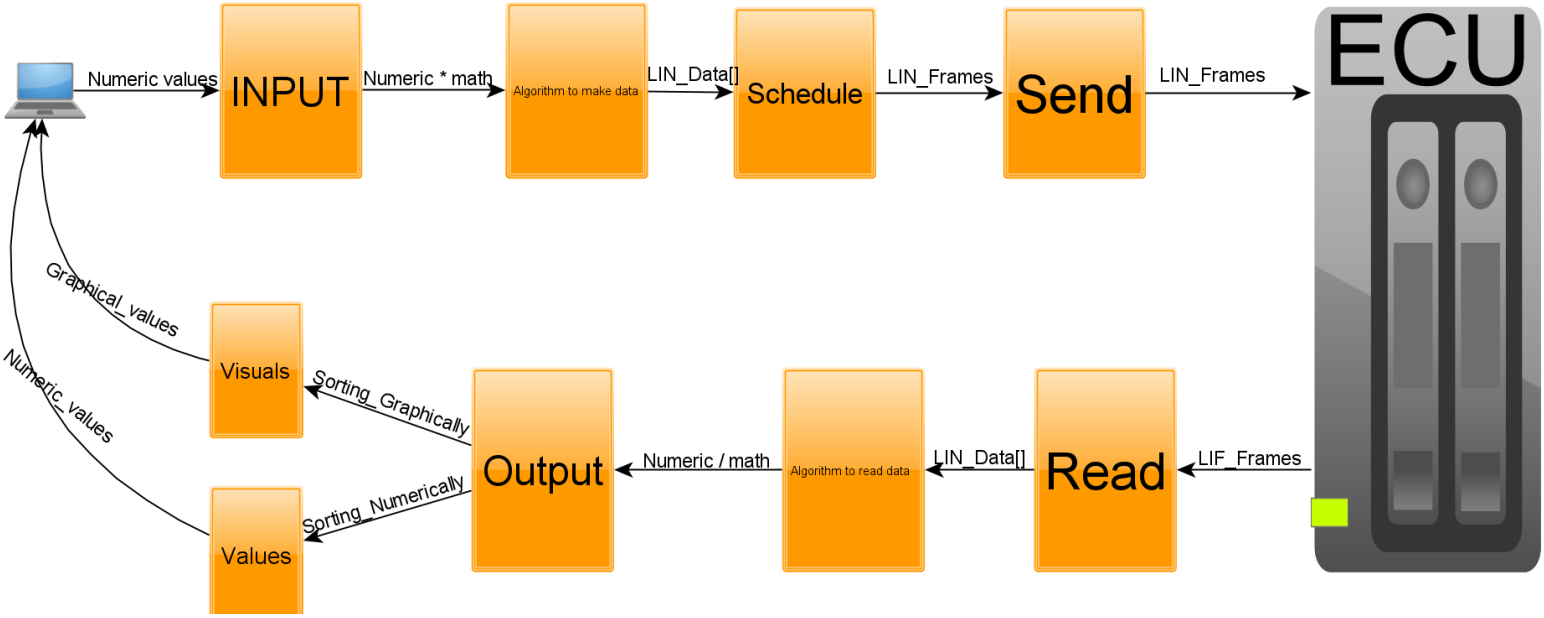
10. ábra architektúra [9]

A szoftver architektúrája az MVC dizájn mintára épül, amint azt az alábbi ábra is mutatja a kezelő használja a felületet, ami egy mögöttes háttérbeli kód lefutása során kiszámolja a szükséges értéket, majd ezzel a kiszámolt értékkel folytatódik a modell, ugyanis minden értéket a Kvaser hardver LIN fordító végre fog hajtani a megadott utasítások szerint. A hardver ezután vissza jelzést ad a modellnek, ami egy frissítésben eltárolódik és ezt feldolgozva megint a háttérben futó kód kiszámolva egy megjelenítést képez, itt gondolni kell numerikusan szám ábrázolásra, és egy grafikus ábrázolásra a felhasználó felé.

4 Integráció

4.1 Szofvter

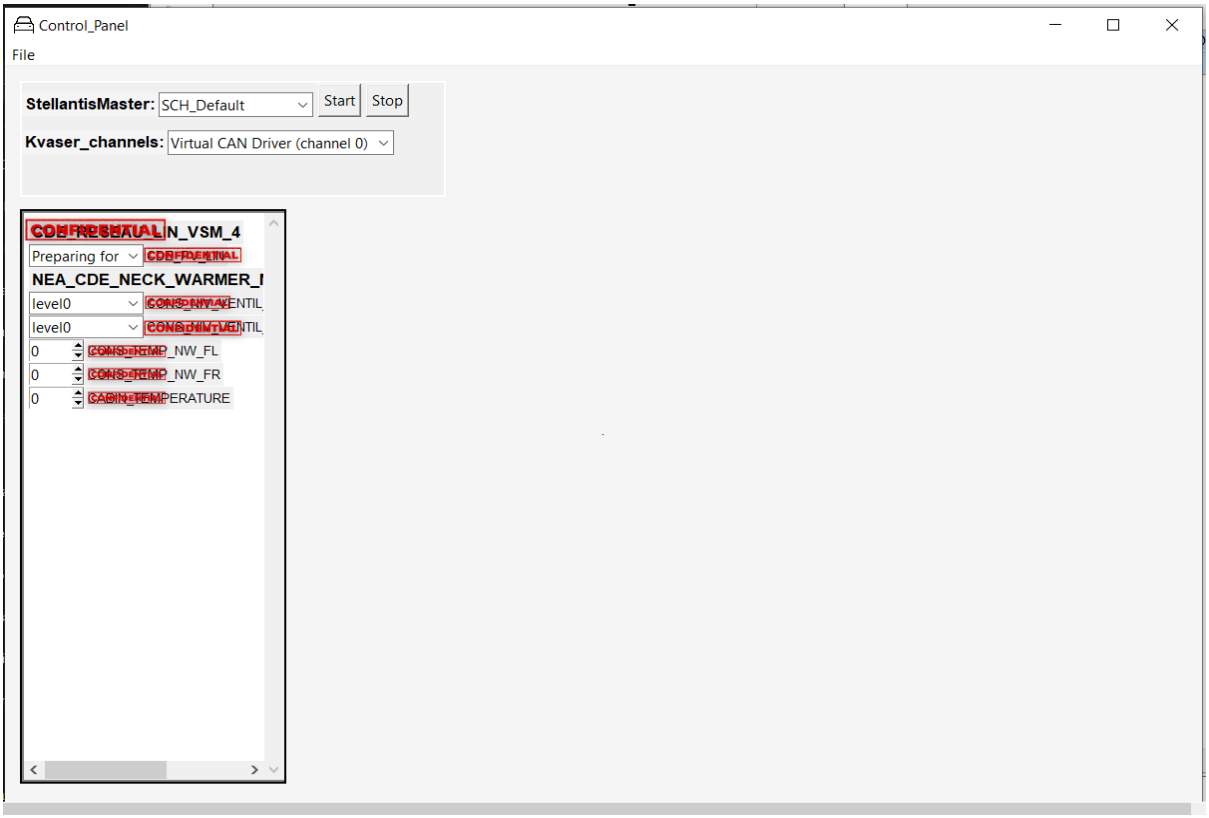
4.1.1 Szoftver Dinamikus lefutása



11. ábra szoftver dinamikus lefutása (saját ábra)

4.1.2 Visszatekintés

Az egyetemi tanulmányaim alapján a fejlesztést a Tkinter nevű keretrendszerben kezdtem el megvalósítani, egy darabig nagyon jól, átláthatóan és magabiztosan haladtam a projekttel, viszont kellő mennyiségű fejlesztés után, mikor már elértem a multithreading-es irányításhoz, és miután sikerült megírni az írás logikáját, illetve kitalálni az alapjait a grafikus megjelenítés formájának, kezdtek átláthatatlanná válni a modulok közötti interakciók.



12. ábra TKinter fejlesztés(fejlesztés (saját ábra)

A képről a jelek neveit szerzői jogok miatt olvashatatlanná tettem.

Az ábrán látható, hogy ez már egy előre haladott állapotban készült kép.

Itt a célom egyharmadánál jártam, ekkorra sajnos már annyira átláthatatlan és bonyolult volt a kód a sok számolás miatt, ahhoz, hogy dinamikusan változzon a keret, illetve mivel hosszú mester keretneveknél lehessen magát keretet elolvasni, hogy mi is van oda írva szükséges volt egy csúszkára, illetve nem csak horizontális, hanem vertikális irányba is szükség volt erre, ugyanis, ha nagyon sok keretet akarunk küldeni és csak az első három fér a képre az nem támogatja a használhatóságot.

Ekkor merült fel az igény egy másik GUI Framework használatára, több ilyen keretrendszer átvizsgálása után találtunk egy sokkal egyszerűbb megoldást a GUI dinamikája megtartására és egyszerűbb kezelésére.

Az új keretrendszerben nem kell továbbiakban számolgatni pixelenként, hogy a különböző grafikai elemek hova kerüljenek és nem kell külön függvényeket írni hogyha esetleg valaki kinagyítja a képernyőt vagy nem olyan konfigurációval használja, mint például én a fejlesztés során.

4.1.3 A PySimpleGUI bevezetése

Az említett új követelményeket a PySimpleGUI könyvtár tökéletesen támogatja, amely egy lényegesen egyszerűbb GUI az előzőhez képest, a további oldalakban a keretrendszer elsajátítása és a program elkészítését fogom bővebben kifejteni.

A PySimpleGUI egy Python könyvtár és keretrendszer a grafikus felhasználói felületek (GUI) készítéséhez. Ezt a keretrendszert arra tervezték, hogy könnyen használható és gyorsan elkészíthető legyen, így ideális választás lehet olyan fejlesztők számára, akiknek nincs tapasztalata a GUI-tervezés terén. Egyszerű és jól dokumentált API-t kínál, amely lehetővé teszi a fejlesztők számára, hogy egyszerűen és hatékonyan hozzanak létre interaktív alkalmazásokat.

Egyik kiemelkedő jellemzője az, hogy több platformon használható, beleértve a Windows, macOS és Linux operációs rendszereket is. Emellett támogat számos különböző „Widgetet”, amelyek segítségével felhasználóbarát és vonzó felhasználói felületeket hozhatunk létre, például gombokat, címkéket, szövegdobozokat, képeket és sok más elemet.

A PySimpleGUI lehetőséget kínál arra is, hogy a fejlesztők testre szabhassák a megjelenést és az elrendezést a projektjük igényeinek megfelelően. Egy egyszerű, szöveges leírás segítségével definiálható a felület struktúrája, majd gondoskodik a megjelenítésről és az eseménykezelésről.

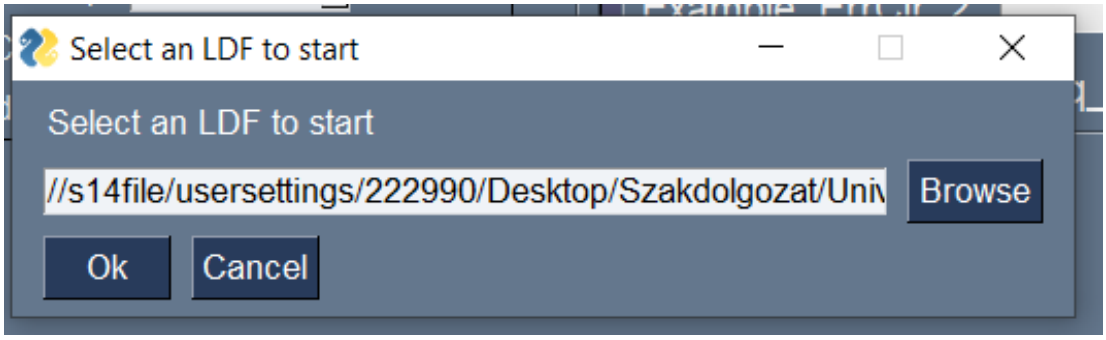
Ennek a könyvtárnak a segítségével könnyedén készíthetünk egyszerű grafikus alkalmazásokat, például számológépeket, feljegyzéseket vagy konfigurációs kezelőket. Az egyszerűsége és hatékonysága miatt egy népszerű választás a gyors prototípusfejlesztésre és az egyszerűbb GUI-projektek megvalósítására Pythonban.

4.1.4 Az első program lefutás

A szoftver dinamikája nem volt megtervezve, a kezdetekben egyszerűen haladtam lépésről lépésre, mivel nem voltam olyan jártas a Pythonban.

A Kvaser irányításához szükséges driverek feltelepítéséhez kitaláltam próbaként egy programot, ami automatikusan feltelepíti a drivereket mindenféle interakció nélkül, amint a felhasználó elindítja az exe fájlt, a program ellenőrzi, hogy az adott driver fel van-e telepítve, ellenkező esetben automatikusan feltelepíti azokat mindenféle felhasználói beavatkozás nélkül a megfelelő helyre, előfordulhat, hogy a driverek telepítéséhez szükséges adminisztrátor jog, ami egy cégnél külön engedélyeztetést vonhat maga után, így az első indításhoz lehetséges, hogy IT segítség szükséges.

Miután minden előkészület megtörtént elkezdődhet a legelső fő lefutás, ami egy fájl útvonal elérésével kezdődik, ami nem más, mint a már a szakdolgozatom elején részletezett LDF fájl elérési útvonala.



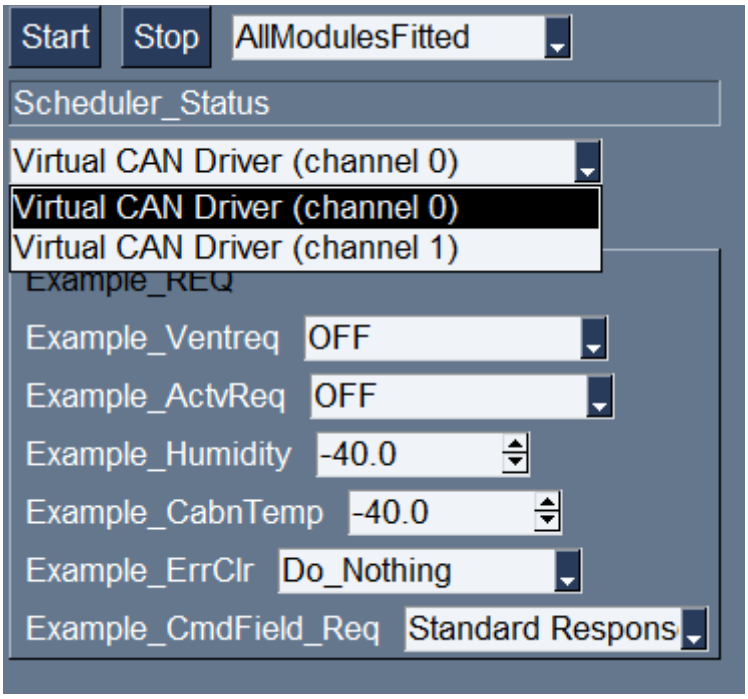
13. ábra LDF útvonal kérés (saját ábra)

Ha ez a megnyitás esetlegesen nem sikeres, a program automatikusan kilép egy hibaüzenet után, ha esetleg az elérési útvonallal vagy az LDF-fel volt probléma, mindenről részletesen tájékoztatást kapunk egy felugró ablakban, mielőtt a program kilépne.

Ha az LDF fájlt kijavítottuk vagy a fájl helyének megkeresésével az elérési útvonalat korrigáljuk, akkor ezekután a programot újra le kell futtatni.

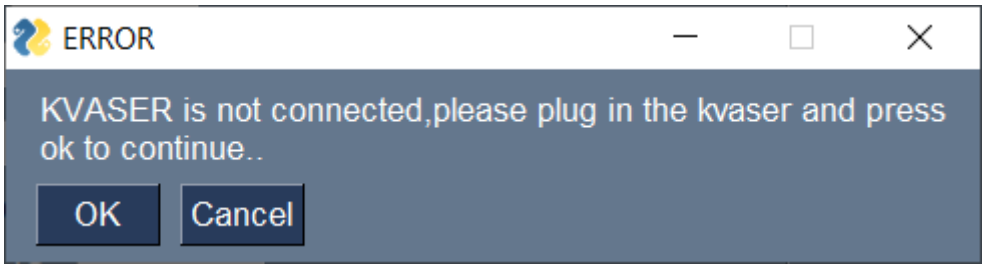
4.1.5 Az LDF feldolgozása

Sikeres megnyitás után az LDF feldolgozásra kerül egy könyvtár segítségével, ennek a segítségével a következő lépés a GUI létrehozása volt. A kezelőfelület az elérhető Kvaser driverekkel kezdődik, ezt megjelenítem egy „combobox” -al, ami gördülő menüs választást tesz lehetővé, itt ki lehet választani melyik csatornát szeretnénk használni.



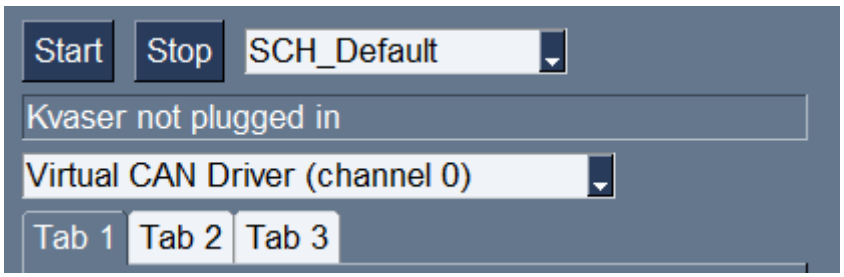
14. ábra Csatorna választás és státusz bár (saját ábra)

Ez a menü induláskor automatikusan megkeresi az elérhető Kvaser drivereket, amelyek elérhetőek. Ha esetleg kettő, vagy annál kevesebb csatorna van jelen a kezdéskor, egy hiba üzenet fog megjelenni a felhasználó számára, hogy nem található megfelelő mennyiségű csatorna, aminek a lehetséges oka lehet az, hogy a hardver nincs ellátva megfelelő tápfeszültséggel, vagy esetlegesen nincs is csatlakoztatva a felhasználó számítógépéhez. Ez a következő lefutáskor újra frissül és újra ki kell választani a megfelelő csatornát. Az előző oldalon található ábra lefutásakor látható, hogy csak 2 csatorna elérhető, tehát a felhasználó nem csatlakoztatta, vagy nem látta el megfelelően a hardvert feszültséggel és így ezt a hiba üzenetet fogja kapni:



15. ábra Hardver nem található (saját ábra)

Rögtön felette található egy „Status Bar”, ami visszajelzést ad az aktuális állapotról, esetleges hibáról, ami fenn állt és már le nyugtáztuk a hiba ablakot. Itt mindig a legutolsó hiba lesz visszaolvasható, ami segítséget ad a felhasználó számára az esetleges kisebb hibák kijavítására, majd újra próbálkozás esetén frissül, információt adva az aktuális státuszról.



16. ábra Status Bar (saját ábra)

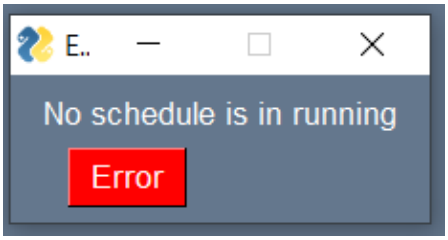
A „Status Bar” felett található "Schedule Table” választó legördülő menü, amiben lehet választani az LDF-ben szereplő ütemező táblázatokból. Ezt már részleteztem a szakdolgozatom elején, ebben vannak a különböző keretek leírva, elképzelhető, hogy éppen csak egy diagnosztikát szeretnénk tesztelni, ez a választó fül erre ad lehetőséget. Esetleg egy továbbfejlesztési lehetőséget megemlítenék itt: a jövőben elképzelhető számomra, hogy ezeket személyre szabottan tudjuk szerkeszteni anélkül, hogy külön megnyitnánk az LDF fájl-t, átírnánk, és újra lefuttatnánk a programot.

A „Schedule Table” mellett található egy Start és egy Stop gomb, ami a küldési ciklust kezdi majd el, egy külön CPU szálon, ugyanis a GUI-nak futnia kell, válaszoláshoz viszont egy teljesen más időzítéssel kell az ütemezést elintézni. Ez a ciklus használja az LDF-ben található ütemezési időt, ami megadja, hogy a LIN-en az adott keretek milyen gyakorisággal, időközönként jelennek meg.

A Start gomb megnyomásával, ha sikeres a küldés és folyamatban van az ütemezés a státusz megváltozik. Ha éppen látjuk a státuszbáron, hogy az ütemezés folyamatban van, a Stop gomb használatával megtudjuk ezt állítani és a státusz automatikusan frissül, ha ez sikeres volt.

17. ábra Ütemezés folyamatban (saját ábra)

A gombokhoz ugyanúgy tartozik hibakezelés is, mint mondjuk a fájl feldolgozáshoz. Példaként, ha esetleg nem fut egy ütemezés sem és megpróbáljuk leállítani, egy felugró ablak tájékoztat minket erről, ezt tudomásul véve az „Error” gombra nyomva lenyugtázhatjuk, a hiba kijavítása után ez nem fog megjelenni.



18. ábra hibakezelés (saját ábra)

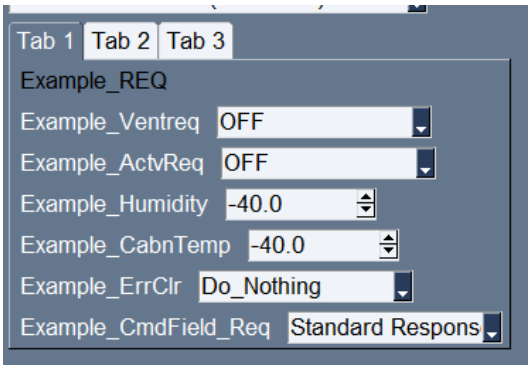
Ezután következik a már kiválasztott ütemező alapján generált küldhető Mester kereteket és jeleket tartalmazó felület. Egyelőre a kód úgy van megírva, hogy az alapértelmezett ütemező értékeket vegye figyelembe, de például a Tkinteres megoldásban csak azokat a jeleket lehetett állítani, amelyek éppen jelen voltak a buszon. Ez a megoldás felhasználó barát megoldás miatt szükséges, hiszen így csak azokat a jeleket képes állítani a felhasználó, ami épp a buszon kint van.

Minden egyes kiküldhető Mester keret neve alá került az összes jel, ami az adott kereten belül található. Ezekhez rendeltem 1-1 bemeneti felületet, ami attól függően változott, hogy van-e megfelelő kódolás a jelhez rendelve és ennek a logikai értékekét dolgoztam fel a későbbiekben.

Ha egy jelhez reprezentatív logikai jelek vannak rendelve, akkor legördülő menüt választottam, hiszen itt sokkal jobban értelmezhetőbb, ha egy készenléti felirat van, mintsem egy 0-s szám.

A fizikai értékekhez egy „Spinboxot” választottam, ami egy bemeneti értéket tartalmazó téglalap, aminek az értékét meg lehet adni manuálisan is, de a lépkedő nyilakkal is lehet értéket választani.

A kezdő értékeket is megadtam, ezek pedig vagy az inicializált értékek, ha van ilyen, vagy pedig a minimum értéket választottam mindenhová.



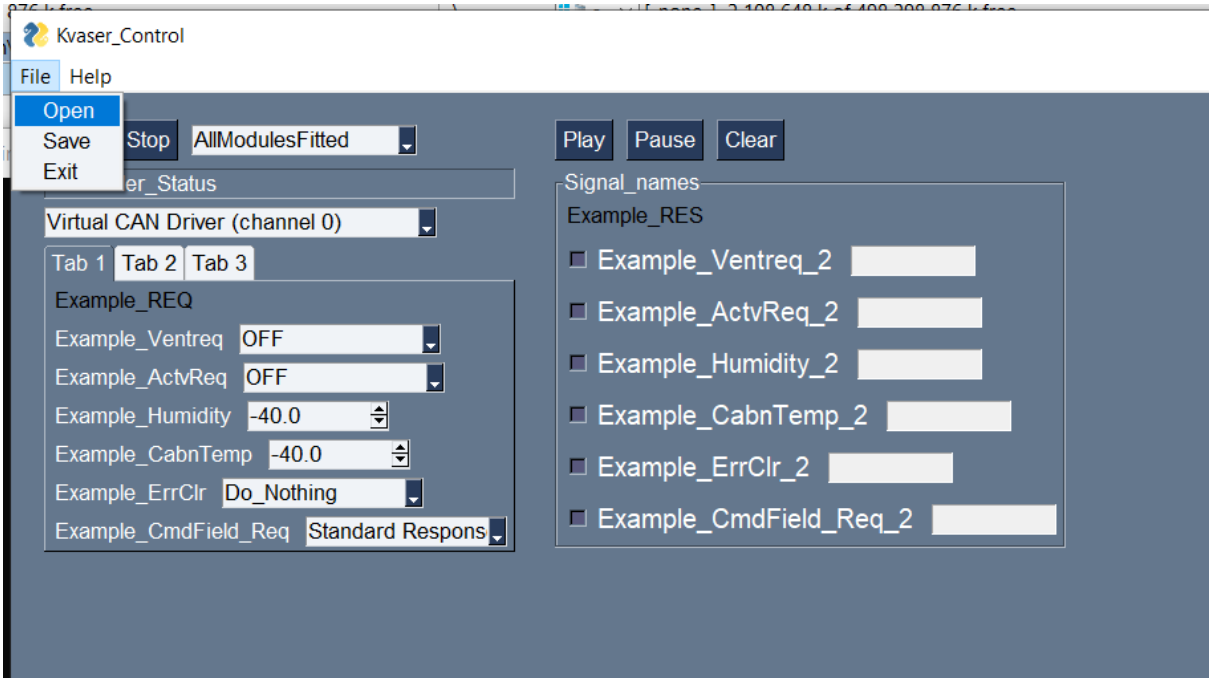
19. ábra vezérlő Tab (saját ábra)

A lépés mértéke a „Spinbox” -oknál nyilván a felbontástól függenek és az adat típusától is. Itt látható, hogy egy lebegőpontos változót is lehet állítani viszont ez bonyolultabbá teszi a LIN-en való számolást.

A többi „Tab” a későbbi fejlesztések miatt került fel. Ezeken egyelőre semmi nem látható.

4.1.6 A Menü bemutatása

Ha sikeres volt az LDF megnyitás és feldolgozás, egy regiszterbe menti az elérési útvonalat, ami a későbbi megnyitásoknál lesz fontos, ugyanis újra megnyitva a programot nem kell ismét beilleszteni az elérési útvonalat, így a program az előző LDF-fel fog lefutni. Amennyiben mégis szeretnénk másik LDF-fet használni, mivel más projektet akarunk esetlegesen tesztelni, mert egy projekt véget ért, vagy esetleg máson is dolgozik egy fejlesztő, erre is van lehetőség a Menü fülön keresztül az „Open” opció választásával ezt az elérési útvonalat frissíteni.



20. ábra Menu-bar (saját ábra)

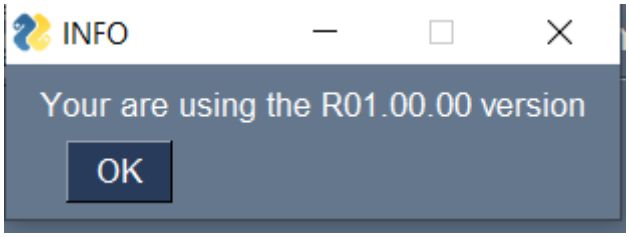
A menü egyelőre elég kezdetleges, majd később bővül további funkciókkal.

Az „Open” parancsot választva újra felugrik a fájl útvonalát kérő ablak, amit az előbbiekben az első futtatásnál már használtunk, és ugyanez a hibakezelési procedúra következik.

A „Save” -gombbal lehet későbbi fejlesztések során elmenteni többféle projektet, hogy ne kelljen mindig minden LDF-et egyesével beolvasni.

Az „Exit” opció kilépésre szolgál.

A „Help” gomb pedig csak visszajelzést ad az aktuális software verzióról, ez segítséget nyújthat akár hibakezelésnél, akár nekünk fejlesztési lehetőségre, ha valaki bármit talál a programban, ami nem ideális akkor tudja jelezni felénk, például egy emailben leírva, hogy baj van az „R1.00.00” -ás szoftverrel.



21. ábra szoftver verzió (saját ábra)

4.1.7 A küldés bemutatása

Mindez a rész a megjelenítés része volt, most áttérnék kicsit a küldés részre.

Minden eddig elhangzott fejlesztés szükséges volt a küldéséhez, viszont ami ezután jött az volt igazán a dolgoknak a neheze, ugyanis fel kellett az összes itt lévő adatot dolgozni, a megfelelő jelek értékeit a GUI-ról megfelelő jelekhez kellett hozzárendelni, és a buszra olyan formában kitenni a kereteket, hogy a jelek meghatározott start bittel rendelkeznek és hosszúsággal, ennek a problémának a megoldására létrehoztam egy algoritmust, ami csinál egy „dictionary”-t, ami tartalmazza a keretek „ID”-ját kulcsként, ami szükséges a buszon lévő üzenetek azonosítására. A kulcs megadásával tudjuk megkapni az adatot ebből a dictionary-ból, ami tartalmazza a kezdő bitet, illetve a hosszúságát az összes jelnek az adott kereten belül.

Foglalkozni kellett mindezek után miután társítottuk a megfelelő jeleket a megfelelő keretekhez azzal is, hogy ha egy 1 byte-os adatot szeretnénk a buszra tenni, és ez az adat a keret 1. bájt második felén van, és a 2. bájt első felén, ezekre az elehetőségekre kifejlesztettünk egy algoritmust és nem csak 8 bites adatokra, hanem 16 bites jelekre is megírtuk ugyanezt a logikát követve, küldés során a felnem használt biteket pedig 0xFF-fel töltöttem ki, ahogy azt a szabvány is előírja.

Bitműveletek használata: Vegyük észre, hogy egy bájt 256 lehetséges értéket vehet fel (0-tól 255-ig). Ha egy bájt értékét két 2-bites részre akarjuk szakítani, akkor a lehetséges értékek számát csökkentenünk kell. Például a 0-3 értékek számára (2 bites), a bájt első két bitjének 00-nek vagy 01-nek kell lennie. A maradék 6 bit az érték része lehet.

Bitműveletek használata a maradékértékre: Az előző lépésben kiválasztott első 2 bitet elvontuk, és most az új bájtnak (a maradéknak) csak a maradék 6 bitjét vizsgáljuk. Így további szűréseket alkalmazhatunk, például megnézhetjük, hogy az első és második bit eltér-e. Például, ha az első bit 0 és a második bit 1, akkor az egyik 2-bites rész, ha mindkét bit 1, akkor a másik 2-bites rész.

1. **Programozási példa:**

Egy példa a Python nyelven, ahol a byte változót két 2-bites részre osztjuk:

```
def split_byte(byte_value):
    first_two_bits = (byte_value >> 6) & 0b11  # Első 2 bit
    remaining_six_bits = byte_value & 0b00111111  # Maradék 6 bit
    return first_two_bits, remaining_six_bits
```

2. **Példa használat**

```
input_byte = 0b11011010
result = split_byte(input_byte)
print(f'Eredeti bájt: {input_byte:08b}')
print(f'Első 2 bit: {result[0]:02b}')
print(f'Maradék 6 bit: {result[1]:06b}')
```

3. **Példa eredmény:**

Eredeti bájt: 11011010

Első 2 bit: 11

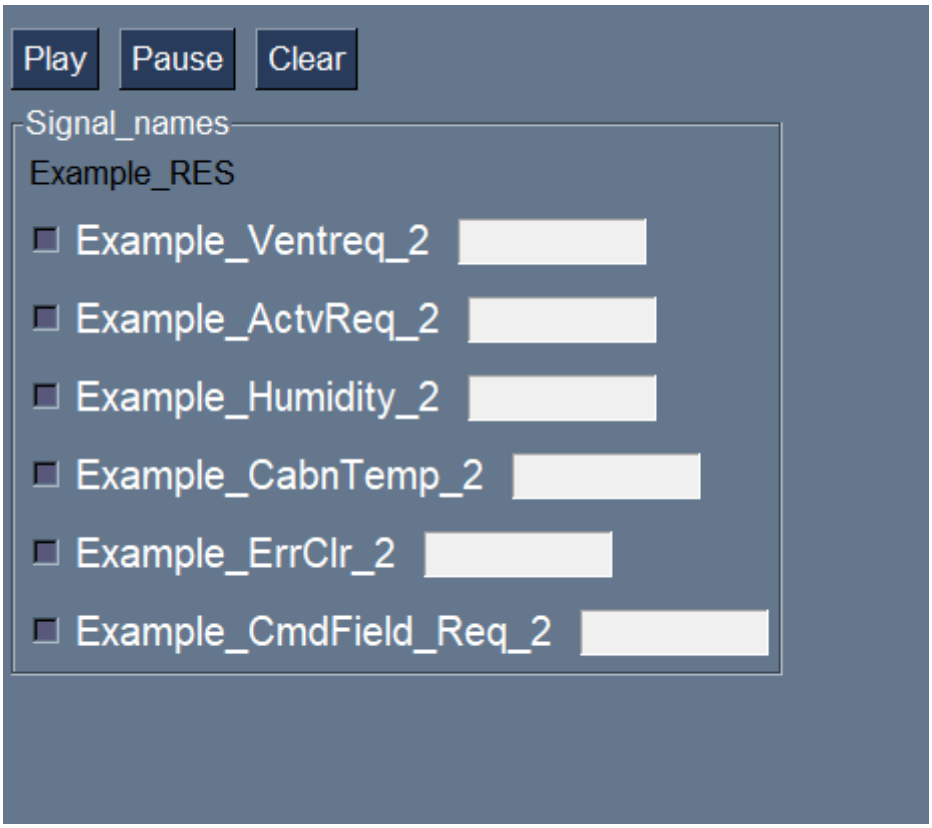
Maradék 6 bit: 011010

Az itt látható algoritmus csak egy példa algoritmus, a saját algoritmust szerzői jogok miatt nem illesztettem bele a szakdolgozatomban, viszont ennek az algoritmusnak fontos szerepe volt az egészben szinte ez vitte el a legtöbb időt a fejlesztésben, de ez felhasználható volt később a visszaolvasás során is.

4.1.8 A visszaolvasás

Középre a visszaolvasott jelek kerültek ugyanazzal a logikával, mint a küldésnél is minden keret alá a megfelelő jel került, a kerethez tartozó jelek a megfelelő keret alá kerültek, ezek mellé került 1-1 „checkbox”, kis négyzet, ami választásra ad lehetőséget, egyszerre többet is lehetséges kiválasztani.

Kiválasztás során egy kis pipa jelenik meg a négyzetben, és egy téglalap is került a jelek mellé egyfajta reprezentatív jelleggel, ide kerülnek a visszaolvasott értékek numerikusan, a „checkbox” -ra, a későbbiekben lesz szükséges grafikus ábrázoláshoz, a visszaolvasáshoz a szakdolgozatomban említett algoritmust kellett használni, csak pont a fordított sorrendben, hiszen itt a buszon lévő üzenetekből kellett vissza fejteni az értékeket.



22. ábra buszon szereplő jelek olvasása (saját ábra)

4.1.9 Grafikus megjelenítés

A fogadás táblázata fölé került 3 db gomb is ezek a grafikus megjelenítéshez szükségesek.

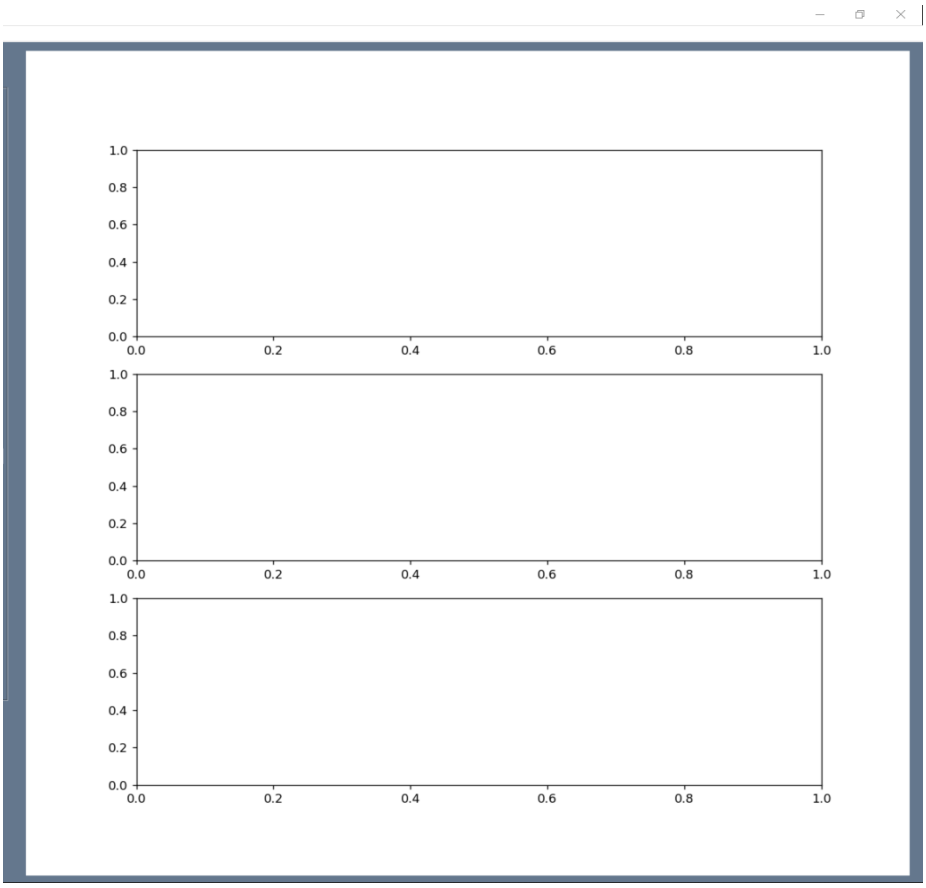
23. ábra Grafikus megjelenítés gombok (saját ábra)

Miután kiválasztottuk melyik jelet vagy jeleket szeretnénk megjeleníteni a checkboxok segítségével, automatikusan létrejön egy plot, amit a „Play” gomb megnyomásával egy thread elindulásával automatikusan másodpercenként frissíti az adott plotot.

Egyelőre csak a bejövő üzeneteket lehet megjeleníteni, de fontos lesz a továbbiakban egymás alá tenni például egy kimenő üzenetet és egy bejövőt, és ezeknek az idejét összehasonlítani, hogy például az adott eszköz milyen gyorsan reagál az adott üzenetre, vagy esetleg figyelmen kívül hagyja másodpercekig, vagy már akár 100-ms alatt is képes feldolgozni azt az üzenetet és képes akár válaszolni is.

Ami még fontos lenne itt kiegészítésnek egy „spinbox” -al, amivel lehetne változtatni a mintavételezési frekvenciát, mert jelenlegi állapotában a kód beégetett 1 másodpercet tartalmaz.

A plot segítségével akár több jelet is ábrázolhatunk grafikusan, itt egy példa 3 db checkbox kiválasztásakor létrejött plotokra:



24. ábra 3db jel üres grafikonja (saját ábra)

A „Stop” gomb megnyomásával a grafikus ábrázolás megállítható és megnézhetjük, elemezhetjük az adott grafikus értékeket, hogy a fejlesztett ECU megfelel-e az elvárt működésnek.

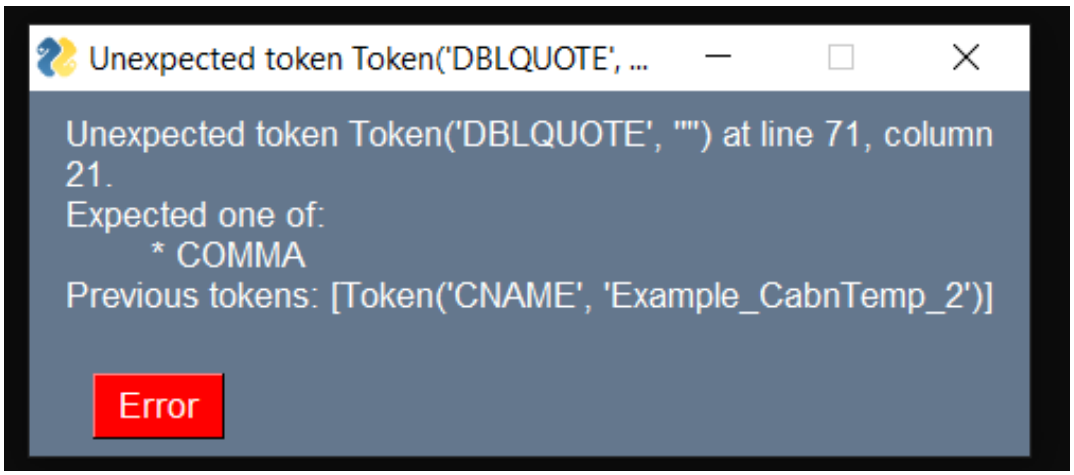
A „Clear” gomb megnyomásával a jelenlegi ábrázolt grafikonokat törölhetjük, ha már nem szeretnénk ezeket az adatokat használni, illetve, ha már nagyon sok ideje futtatjuk a szimulációt előfordulhat, hogy olvashatatlanná válik a grafikon, így erre ad megoldást ez a gomb.

A fejlesztés végezetével generáltam egy exe fájl-t, amit bárki elindíthat a saját gépén, és szabadon használhatja az eddig bemutatott programot a további fejlesztéseken a későbbiekben fogok dolgozni és remélhetőleg a mesterszakon folytatott tanulmányaim során kiegészítem minden említett extrával.

5 Tesztelés

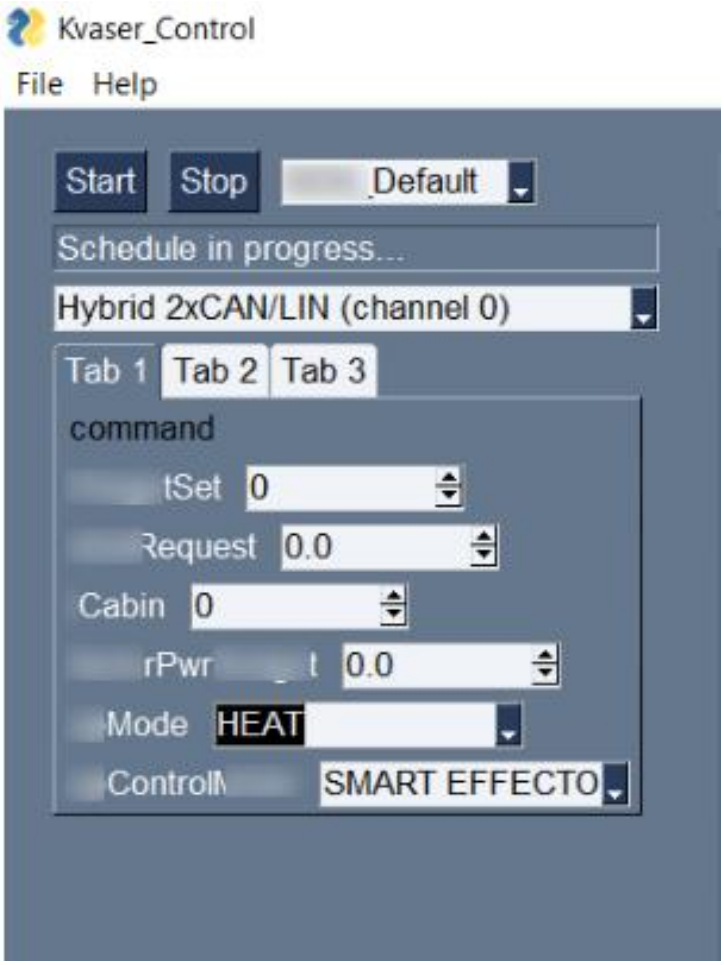
A fejlesztés alatt folyamatos tesztelésen esett át a program, amivel képes voltam több és több hibakezelést beiktatni a rendszerbe, hogy minél egyértelműbben lehessen minél kevesebb tudással működtetni a programot és ezáltal felhasználó barát legyen.

A tesztelést úgy is próbáltam százszázalékosra tökéletesíteni, hogy minden lehetséges variációt kipróbáltam, mint például rossz LDF fájl használása:



25. ábra rossz LDF fájl (saját ábra)

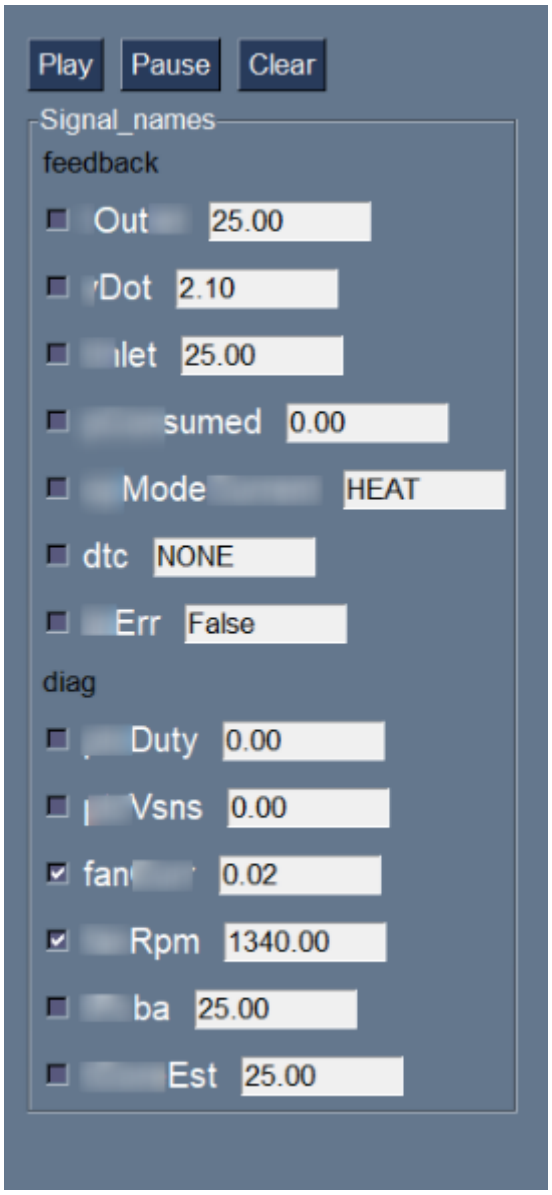
A kódban nyilván kivételkezelővel ezeket lekezelve a felhasználó felé megjelenítve nagyon könnyen ki lehet javítani az aktuális hibákat. Nagyobb problémákról, illetve hozzá nem értő személyek számára bele kell rakni egy fejlesztést, ami egy esetleges log fájlba lementi a hibaüzeneteket, hogy aztán emailben elküldje nekünk, és egyfajta segítő vonalat kialakítva segíthessünk a program megfelelő futtatásáról.



26. ábra Kontroll jelek (saját ábra)

Tesztelés során használt jeleket szerzői jogok miatt kiretusáltam.

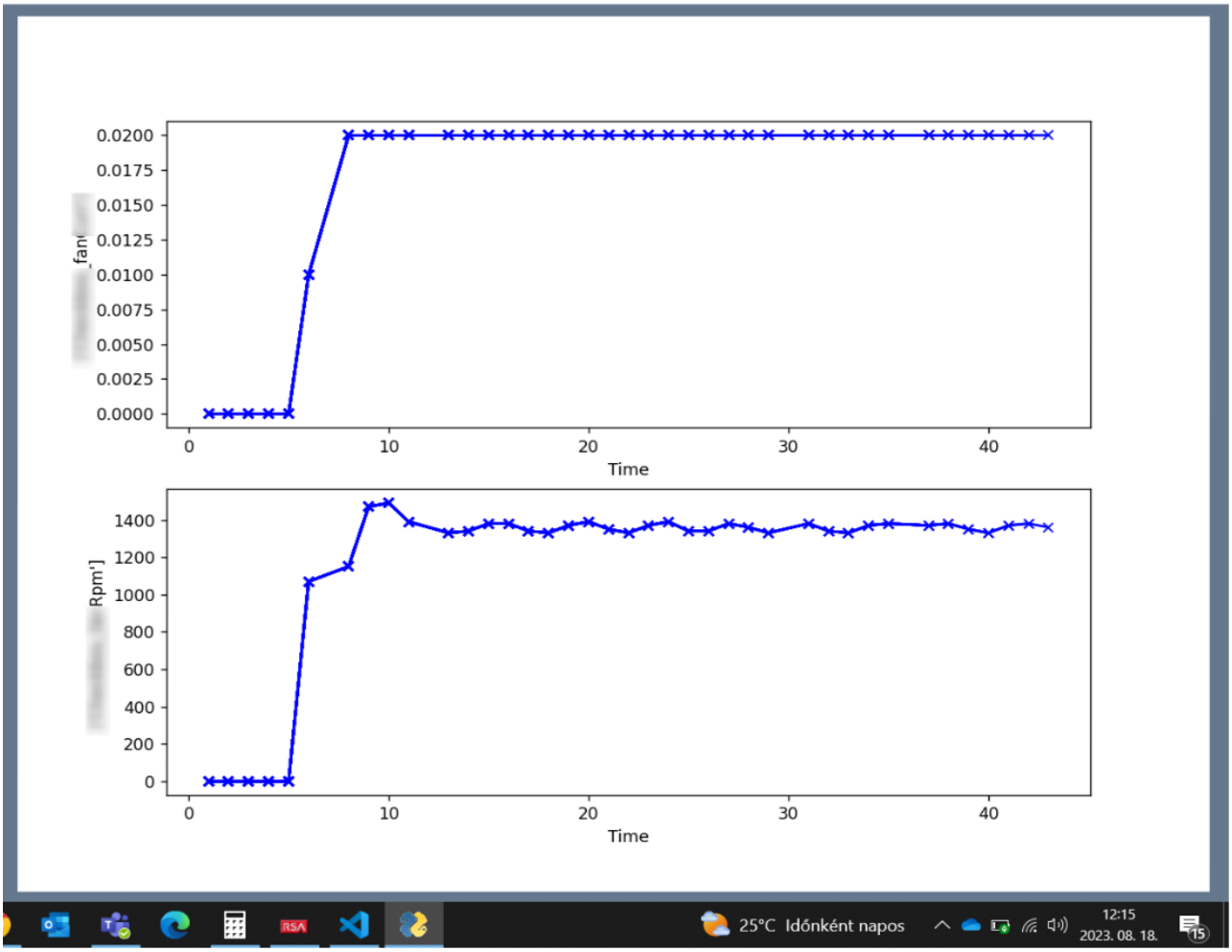
Az itt látható jelek segítségével egy meglévő Slave-ECU-val, annak Mester jeleit szimulálva sikerült valós idejű kommunikációt folytatnom. Látható a szakdolgozatomban részletesen leírt módon kiválasztottam az összes jelet, a megfelelő csatornát a hardveren és a nekem szükséges ütemezési táblát, beírtam az adott értékeket, illetve kiválasztottam, hogy milyen módba kapcsoljon az adott ECU, majd elindítottam az ütemezést, amiről visszajelzést kaptam az adott státusz fülön.



27. ábra Jelek visszaolvasás teszteléskor (saját ábra)

A képről a jelek neveit szerzői jogok miatt olvashatatlanná tettem.

Az előbb elküldött ütemezés alapján a képen látható példa az egyik saját fejlesztésű ECU-ról való kommunikáció során vissza érkező üzeneteket jelenítem meg egy-egy dobozban. A jelölőnégyzetekre a következő kép miatt vannak bepipálva.



28. ábra grafikus ábrázolás a bejövő jelekről (saját ábra)

A képről a jelek neveit szerzői jogok miatt olvashatatlanná tettem.

Ezen az ábrán látható a grafikus megjelenítés az előbb kiválasztott jelek alapján, másodpercenként ábrázolva.

A tesztelést során a dolgozatomat többféle projektünkön is kipróbáltam számos eszközzel képes voltam kommunikálni hiszen a program egy mester modult képes szimulálni így kimondható volt, hogy a program maga működik.

6 Tovább fejlesztési lehetőségek

Rengeteg tovább fejlesztési lehetőséget nyit ez a projekt példaképpen felsorolnék néhány lehetőséget:

- UDS (ISO 14229 Szabvány) üzenetek implementálását,
- CAN busz kommunikáció teljes kialakítása hasonló platformon,
- Lin megfelelés teszt össze állítása,
- Bootloader flashelési lehetőség,
- Loggolás a LIN adatokról,
- Trace kialakítása,
- Üzenet/üzenetek időközbeni beszúrása,
- A trace és grafikon konfigurálása,
- Bármiféle teszt automatizálási lehetőség akár egy több órás fűtésbekapcsolás és visszajelzés figyelés, hogy működik-e az eszköz,
- Több vizuális elem beillesztése akár LED-ek akár analóg órák.

7 Összegzés

7.1 Összegfoglalás

Összefoglalásként elmondható, hogy rengeteg időt eltöltöttem a fejlesztés részével tényleges architektúra nélkül, ami mára nyilvánvalóvá vált, hogy elengedhetetlen az ilyesfajta fejlesztéshez, előbb azt kellett volna elkészíteni és a kódolás része utána sokkal egyszerűbb lett volna és átláthatóbb.

A további fejlesztés során ezt fogjuk követni és készíteni fogunk egy architektúrát, hogy átlátható legyen és optimalizáltabb is a kód.

Mivel most jelenleg csak én tudom nagyjából, hogy mi található a kódban így ha valaki próbál bármilyen fejlesztést csinálni esetleg adatot használni a kódból lehet, hogy olyan dolgokat változtat meg ami tönkre teszi az eddigi munkám, tehát elmondható hogy igazából ez egy DEMO verzió lett.

Ez viszont teljes mértékben ráébresztett minket arra, hogy nem lehetetlen egy ilyen program megírása és a cég is rájött, hogy ezzel rengeteg pénzt tudunk megspórolni a fejlesztés során .

7.2 Summary

In summary, it can be said that I spent a considerable amount of time on the development without a proper architecture, which has now become evident as essential for such development.

It should have been created beforehand, and the coding part would have been much simpler and more transparent afterwards. In the upcoming development, we will follow this approach and create an architecture to make the code more transparent and optimized.

Currently, only I have a rough understanding of what is in the code, so if someone tries to make any developments or use data from the code, they may inadvertently change things that could disrupt my previous work.

Therefore, it can be stated that this is actually a DEMO version that has fully alerted us to the fact that writing such a program is possible. The company has also realized that we can save a significant amount of money during development with this approach.

8 Irodalomjegyzék

[1] https://intrepidcs.com/products/vehicle-network-adapters/neovi-red-2/
[2] https://lipowsky.com/products/Baby-Lin-II
[3] https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn16xx/#c236777
[4] https://www.kvaser.com/product/kvaser-hybrid-2xcan-lin/
[5] https://www.cable-tester.com/rs232-pin-out/
[6] https://www.iso.org/standard/61222.html
[7] https://www.kvaser.com/about-can/can-standards/linbus/
[8] https://microchipdeveloper.com/lin:protocol-app-ldf
[9] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

9 Ábrajegyzék

10 Táblázatjegyzék