

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREŞ
SPECIALIZAREA CALCULATOARE**

**Model funcțional al unui panou de
instrumente de automobil digital cu
mijloace de comunicație industriale CAN**

PROIECT DE DIPLOMĂ

**Coordonator științific:
Dr. Csernáth Géza,
adjunct universitar**

**Absolvent:
Bogdán Norbert**

2016

LUCRARE DE DIPLOMĂ

Coordonator științific:
ș.l. dr. ing. Csernáth Géza

Candidat: **Bogdán Norbert**
Anul absolvirii: **2016**

a) Tema lucrării de licență:

Model funcțional al unui panou de instrumente de automobil digital cu mijloace de comunicație industriale CAN

b) Problemele principale tratate:

- protocoale și magistrale de comunicație în industria auto, interfețe OBD de comunicație
- protocolul și magistrala CAN și aplicațiile sale în industria auto
- panouri de instrumente digitale, principii de funcționare, interfețe, proiectare, implementare
- sisteme cu microcontrolere, proiectare, implementare, programare
- sisteme de comunicație și interfațare fără fir, utilizare, programare
- extragerea și filtrarea și afișarea fluxului de date furnizat de calculatorul de proces al unui autovehicul

c) Desene obligatorii:

- schema bloc a magistralei de comunicație CAN, schema nivelelor de protocol
- schema bloc a sistemului cu microcontroler
- schema bloc și schema detaliată a panoului de instrumente
- organograma programului realizat, rulat pe microcontroler
- schema bloc a sistemului de comunicație

d) Softuri obligatorii:

- algoritm protocol de comunicație serială microcontroler
- algoritm comandă panou de instrumente
- algoritm parser mesaje CAN

e) Bibliografia recomandată:

- [1] Dr. Nagyszokolyai Iván és Dr. Lakatos István. Gépjármű diagnosztika. Typotex, Budapest, 2011.
- [2] Aradi Szilárd. Fedélzeti diagnosztikai protokollok. BME, Budapest, 2013.
- [3] Dr. Fodor Dénes és Dr. Szalay Zsolt. Autóipari kommunikációs rendszerek. Pannon Egyetem, Veszprém, 2014.
- [4] Debnár Péter. Soros kommunikáció a gépjárművekben – a CAN. Budapesti műszaki főiskola, Budapest, 2008.

f) Termene obligatorii de consultații:

- consultații lunare pentru urmărirea evoluției lucrării
- 1.06.2015, 30.06.2015, 21.09.2015, 28.10.2015, 24.11.2015, 27.01.2016, 24.02.2016, 23.03.2016, 27.04.2016, 25.05.2016

g) Locul și durata practicii: Universitatea Sapientia

Primit tema la data de: 31.03.2015

Termen de predare: 27.06.2016

Semnătura Director Departament

Semnătura coordonatorului

**Semnătura responsabilului
programului de studiu**

Semnătura candidatului

Model funcțional al unui panou de instrumente de automobil digital cu mijloace de comunicație industriale CAN

Extras

Introducere

În prezent, creșterea integrării electronice și necesitatea de compactitate a circuitelor electronice a condus la un progres major și în domeniul microcontroalelor. Microcontrollerul a devenit mai mic și mai ieftin, și în aceeași timp și dimensiunile lui s-au redus drastic în ultimele decenii. Din cauza acestor motive integrarea lor în sisteme de comunicație și nu numai a devenit inginerilor din ziua de azi aproape trivial. Această schimbare de paradigmă a adus, de asemenea, o creștere exponențială în dezvoltarea industriei al autovehiculelor. Sistemele electrice din mașini au devenit și ele compacte, eficiente și din ce în ce mai complexe. Automobilele din ziua de azi din punct de vedere electric includ o gamă variată de funcțiuni electronice. Privind aceste funcțiuni putem enumera câteva dintr-e cele mai importante: cele care oferă siguranță și confort, funcțiuni care controlează diferite părți al vehiculului sau cele care asigură încadrarea în normele de mediu. Ca și în cazul tuturor sistemelor digitale electronice, și în aceste sisteme destinate industriei automobilelor, pentru o funcționare robustă și optimă este foarte importantă să nu deranjeze utilizatorul în timpul conducerii. Tocmai din potrivă comunicația între părțile sistemului, cu utilizatorul, dar și cu mediul exterior trebuie să fie promptă.

Informații definitorii al temei

Diagnosticul tehnic este o activitate indispensabilă în industria autovehiculelor. Mașinile utilizate zilnic, sunt supuse unor norme tehnice și legale. Iar periodic obligatoriu automobilul e supus unor măsurători tehnice, în urma căreia se obțin informații esențiale despre starea tehnică a automobilului fără să disturbăm funcționarea a sistemului. Distingem două metode majore de diagnosticare:

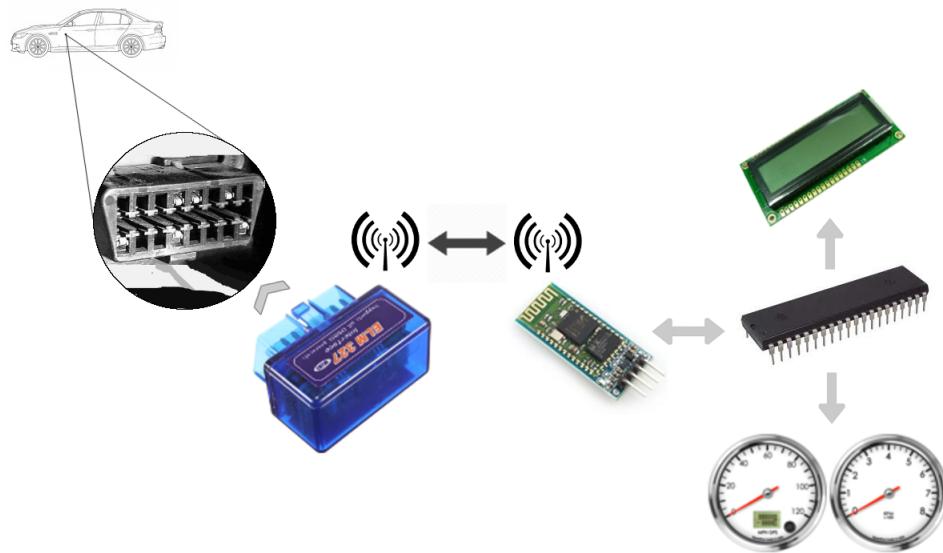
1. Diagnosticul bordului mașinii, e o funcție specifică prin care în mod constant sunt observate prin diferite tipuri de măsurări componentele mașinii și suntem imediat alertați în cazul în care ceva nu funcționează corespunzător. Acesta presupune pentru o diagnosticare optimă a bordului este necesar ca atât partea de hardware și software să facă parte din sistem. În continuare e important de menționat că în cazul diagnostizării bordului acest tip de observație continuă se categorizează în două grupe: Parametrii observați cu o continuitate permanentă (viteză, rotații pe minut al motorului) și cele observate cu o continuitate ocasională.
2. În cazul diagnostizării non-bord, software-ul și hardwerul necesar nu face parte din sistem, deci cumva printr-un mediu acesta trebuie conectate în timpul examinării.

Vehiculele din ziua de azi conține o gamă largă de senzori și faruri, iar numărul lor are o tendință de creștere. Prin natura numeroasă a acestora sistemele de control calsice de tip point-to-point sau centralizat nu a mai putut să facă față. Deci a fost necesar dezvoltarea unui

nou sistem de control distribuit care să fie capabil să funcționeze în gama de baterie, să fie ușor de expandabil, ieftin, să ofere siguranță iar dacă un nod se defectează în sistem asta să nu afecteze integritatea sistemului. În prezent cel mai popular protocol de comunicare folosit în industria auto care satisfac aceste condiții este protocolul CAN. Funcționarea teoretică a protocolului CAN este împărțită în noduri, iar în funcție de prioritatea parametriilor trimise pe magistrala de date, deosebim o magistrală rapidă și una lentă. Sistemul de diagnoză al mașinii este conectat la această magistrală care poate fi accesat cu ajutorul unui circuit de conectare.

Subiectul lucrării

Tema tezei de diplomă, este studierea comunicației în automobile - protocoluri de comunicație- pe de o parte, iar demonstrarea acestuia prin construcția unui sistem bazat pe un microcontroler care e capabil să adune și să proceseze câteva date care descriu funcționarea mașinii. Acest sistem conține două sub sisteme, și anume una mecanică iar una destinat microcontrolerului. Scopul nostru e să și afișăm utilizatorului datele procesate. O parte integrală a proiectului e planificarea și implementarea softwareului care va controla microcontrolerul. Sistemul se va conecta la sistemul de comunicație a mașinii printr-un circuit de conectare de tip ODB, aceste urmărește valorile senzorilor pe care le dorim să procesăm. Sistemul comunică cu automobilul prin semnale Bluetooth.



Arhitectura sistemului și componentele principale

Sistemul realizat poate fi distins în două părți importante: primul este unitatea de control care coordonează bordul meu, și perifericele conectate, iar al doilea parte ar fi circuitul de conectare care realizează comunicația cu sistemul de diagnoză a vehiculului.

Componentele folosite pentru demonstrarea sistemului de diagnoză:

- ELM327 circuit de conectare de diagnoză

- sursă de 5-10 V
- microcontrolerul
- motor incremental
- modul de Bluetooth HC-05
- seonzor Hall
- LCD alfanumeric
- butoane
- port RS232 pentru programare

Sistemul trebuie să conțină următoarele funcționalități:

- programabil prin portul USB cu software de rezidență
- bazând pe funcțiile unui automobil real, indicatorii de viteză și rotații pe minut să funcționează la fel
- bordul construit să fie portabil cu alimentare proprie
- bordul trebuie să conțină o interfață grafică pentru utilizatori
- meniul de control al bordului și câteva parametrii al mașinii să fie afișate pe LCD
- posibilitatea de navigare prin meniu cu ajutorul butoanelor
- sistemul să comunice cu sistemul de diagnoză a mașinii prin semnale bluetooth

Construcția, implementarea și punerea în aplicație a sistemului

Cum am mai menționat mai sus, sistemul se compune din sub sistemul mecanic și cel electronic controlat prin software.

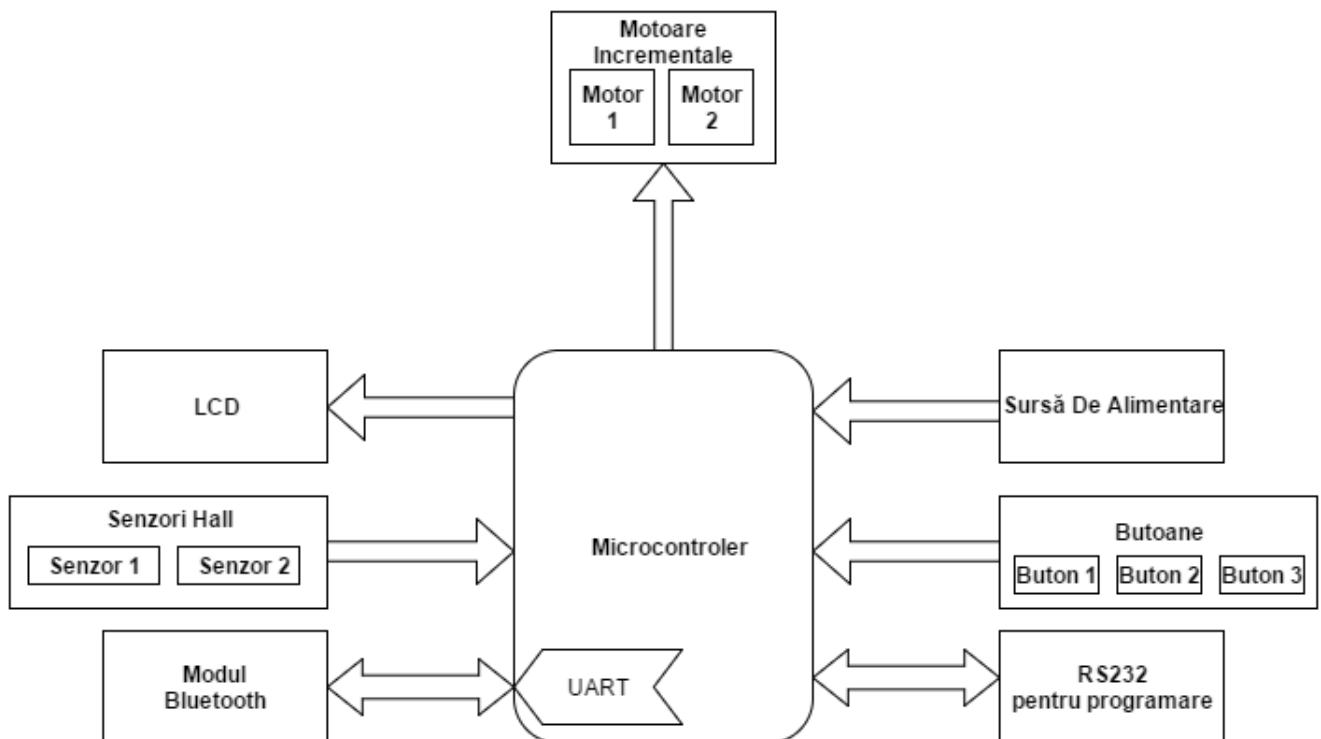
Planificare Hardware

Planificare Mecanică

Baza bordului creat constă dintr-o foaie de plastic cu dimensiunile 24.5 x 21.5 cm, pe aceasta sunt fixați aparatelor de afișare a măsurătorilor, cadrul de suport pentru LCD, precum și circuitele electronice necesare. Instrumentele de afișarea a măsurătorilor (viteză metru și instrumentul de măsurarea rotației pe minut al motorului), se realizează prin motoare incrementale care sunt fixate la cadru cu ajutorul cătorva panouri de plastic formate să se îmbine cu cadrul. Pe o parte a panoului se află motoarele și pe cealaltă parte scara pentru măsurarea vitezei și a rotației pe minut. Panoul e fixat de cadrul iar între cadrul și panou este fixat afișajul alfa-numeric care e lipit de cadrul.

Planificare Electronică

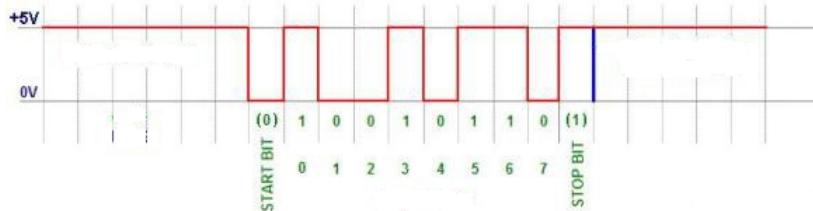
Sistemul electronic al bordului sincronizează și controlează intregul sistem. Folosind cunoștințele dobândite la cursurile de Microcontrolere și Electronică analogică am planificat circuitul electronic. Pentru proiectare să folosesc programul Proteus 8. Primul pas în proiectarea sub sistemului electronic a fost alegerea unui microcontroller cu platformă de dezvoltare potrivit. Platforma de dezvoltare aleasă se bazează pe familia de microcontrolere PIC, din această cauză am ales microcontrollerul de tip 4620 din familia PIC18F.



Principalele aspecte la alegerea microcontrollerului au fost: să aibă periferie de comunicație serială tip UART, porturi I/O, cel puțin două circuite de timer. Microcontrollerul PIC18F4620 are arhitectură bazată pe 8 biți, cu circuiți de timer bazată pe 8 și 16 biți, iar în ultimul rând are integrat o memorie pentru programe de tip flash de dimensiunea 64KB. Pentru funcționare unitatea centrală de control are la dispoziție un oscilator extern de 4 MHz. Aceasta prin softwarul rezident folosind circuitele PLL este multiplicat de 4 ori, în final sistemul are un semnal de ceas de 16 MHz.

Idea generală de funcționare a sistemului este că circuitul de conectare de diagnostizare este conectat la mașină, care primește de la microcontroller semnale de cerere, iar circuitul de diagnostizare trimite înapoi răspunsurile corespunzătoare cererii. Microcontrollerul primind acest răspuns pune în acțiune motoarele incrementale, și trimite semnalul procesat la LCD, astfel rezultatele fiind afișate utilizatorului.

PIC-ul trimite și primește datele prin periferia sa serială, asincronă. Comunicația serială asincronă se realizează prin modulul de Bluetooth, prin aceeași se trimite câte un bit la fiecare semnal de ceas.



Circuitul de conectare de diagnostizare ales e un circuit ELM327 care în principiu realizează conexiunea între circuitul periferic de comunicare serială și sistemul de diagnostizare a mașinii. Circuitul suportă o gamă largă de protocole de comunicație utilizate în industria automobilă. Este implementat relativ la listă lungă de comenzi de configurație care e ușor accesibil. Astfel îl putem configura ușor conform necesităților noastre. Cereriile primite prin canalul serial sunt interpretate de circuitul ELM, și sunt transformate de el conform protocolului de comunicație utilizate de mașină. Iar când primește un răspuns aceeași proces se repetă în sens invers.

Extensia indispensabilă a PIC-ului e periferia de Bluetooth, care în cazul nostru e un modul de tip master-slave HC-05. Prin aceasta se trimit și cere datele, pentru asta trebuie configurat cu comenziile "AT". Acest pas e necesar pentru că după alimentare să se conecteze automat la circuitul ELM327. Modulul e conectat la pinni "RX" și "TX" ale PIC-ului, prin aceste pinuri se realizează comunicația serială, asincronă.

Datele primite de la mașină (viteza și rotațiile pe minut ale motorului) după procesare sunt afișate pe LCD, și vizualizate de către motorul incremental unipolar 28BYJ-48. Aceste motoare sunt controlate de circuitul de control ULN2003. Datele primite în acest caz sunt transformate în semnale digitale, și le trimitem către motoare. Care pe axa lor au un indicator și în față indicatorului au o scală. Astfel simulăm indicatoarele de pe bordul mașiniilor. Pentru a configura poziția de pornire a indicatoarelor sunt folosite niște senzoare de tip Hall, care în poziția zero a indicatorului se opresc, datorită magnetului mic atașat pe vârful indicatorului.

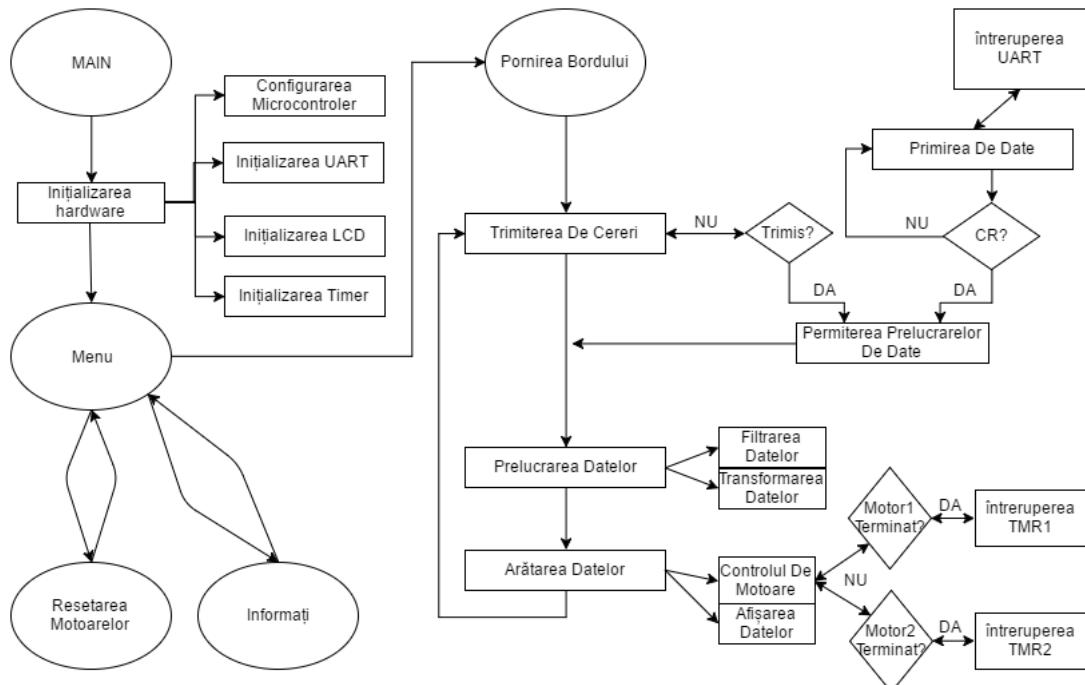
Restul datelor urmărite sunt temperatura lichidului de răcire, și temperatura aerului aspirat sunt afișate utilizatorului pe un modul de LCD alfanumeric cu dimensiuni de 4x20.

Funcționarea sistemului este sincronizată printr-un meniu accesibil prin LCD, și e controlată de butoanele de pe platforma de dezvoltare.

Sistemul în timpul implementării și testării a fost alimentat prin portul USB. Însă am testat și cu o baterie de 9V, cazul în care cu ajutorul unui regulator am redus nivelul de tensiune la tensiunea de funcționare a microcontrollerului adică 5V.

Planificare Software

Software-ul scris pentru configurarea microcontrollerului să realizează în mediul de dezvoltare MPLAB IDE v8.92. Codul să scris în limbajul C, și a fost compilat în instrucțiuni binare cu ajutorul compilatorului C18. Codul de executat ajunge în memoria de programare a PIC-ului prin software-ul rezident de încărcare (bootloader) care comunică cu calculatorul prin periferia de comunicare rs322 printr-un cablu serial de USB.



Protocolul de comunicație dezvoltat

Toată comunicația e bazată pe caractere, datele sunt trimise într-o formă hexadecimale, și sunt primite la fel de către microcontroller. Este extrem de important, ca PIC-ul când trimite o cerere către mașină să fie adăugat la sfârșitul mesajului un caracter "CR" (Carriage Return), pentru că pe cealaltă parte numai aşa se primește și se procesează mesajul respectiv. Toate byte-urile al mesajului sunt trimise și primite bit după bit de către PIC. Când primim un mesaj de la automobil, se recunoaște prin identificarea simbolului ">" la finalul mesajului. În programul nostru există o funcțiune care are rolul de a opera întreruperile apărute în timpul execuției, iar aceasta caută simbolul mai sus menționat. Pachetul trimis conține tipul cererii, iar codul PID al parametrului dorit. Toate codurile se găsesc pe pagina de web [17]. Pachetul primit conține un byte care semnalizează dacă cerearea a fost servit cu succes (0x41). Urmează codul PID trimis și în pachetul de cerere având o mărime de un byte, iar ultimul e data folosită pe care-l putem trimite mai departe la procesul de afișare. Ultima parte poate avea o mărime de unu sau doi byte, depinde de valorile decimale.

La pornirea programului, în primul pas, se configerează și se initializează întregul sistem. Sunt setate direcțiile porturilor de periferie, periferia UART, timerele și se execută rutinul care pornește afișajul inițial al LCD-ului. Această parte a programului se execută o singură dată la pornire, pe urmă aplicație continuă la meniul implementat. Pe urmă se execută funcțiunea care afișează meniul principal pe LCD, și un ciclu infinit așteaptă la apăsarea butoanelor, pentru a duce execuția mai departe. Dacă se apăsa un buton, se intrerupe ciclul, și se cheamă funcția scrisă pentru punctul din meniu care a fost selectat.

Cea mai importantă funcțiune e aşa numita StartBoard(), care initializează începutul comunicației cu mașina, și controlează și afișează datele primite. După pornirea bordului se trimit unul după altul parametrii sunt în legătură cu cererile, iar după primire, cu ajutorul funcțiunii care analizează datele primite sunt afișate. Primirea datelor este operat de către un întrerupător de prioritate mare, iar controlul motoarelor incrementale sunt operate de către doi timeri care sunt implementate în întrerupători de prioritate mică.

Testarea sistemului

Testarea sistemului a constat din doi pași. În primul pas am testat comunicația serială UART, acesta fiind o parte esențială a sistemului. Pentru testarea am folosit diferite programe de tip terminal, folosind aceste programe am putut verifica corectitudinea datelor trimise și pînă prin UART, iar la fel am verificat și partea de controler dacă datele nu au suferit modificări.

În al doilea pas am testat sistemul construit cu un automobil. După ce am introdus circuitul de conectare în portul de diagnostizare a mașinii, am pornit mașina și bordul construit, iar în final am pornit aplicația de pe microcontroller. Pe figura de mai jos se poate observa că valorile măsurate de bordul construit, în timp real, corespund cu valorile de pe bordul mașinii. Indicatoarele mele arată aproximativ aceeși valori, iar pe LCD putem citi valorile pe care le vedem și pe bordul mașinii.

Concluzii

Prin realizarea tezei de licență am reușit să mă obijnuesc cu protocoalele de comunicație utilizate în industria automobilelor. Am clarificat principiile de funcționare a diagnostizării tehnice, observând astfel cum se derulează zi de zi funcționarea senzorilor într-o mașină. Alegând această temă am dobândit experiență în domeniul planificării unui aplicații de microcontroller folosit în lumea reală. Urmărind pașii execuției proiectului am reușit să imitez și să rezolv execuțiile tehnice pe care le poate întâlni un student de calculatoare (sau automatizări) în piața muncii.

Am reușit să construiesc un sistem cu care am dovedit principiile de funcționare a diagnostizării. Mai exact cu un sistem bazat pe un microcontroller, printr-un circuit de diagnoză pot să mă conectez la sistemul de diagnoză a automobilului. Pot să cer date tehnice de la mașină, și să le afișez unui utilizator. Funcționalitatea acestui proces a presupus stabilirea unui mediu de comunicație foarte promptă, pe care l-am realizat prin protocolul Bluetooth. Circuitul electronic creat conține pe lângă microcontroller, și un model simplificat al bordului mașinii a fost creat să fie portabil, și relativ simplu de testat la majoritatea mașinilor moderne.

Sistemul are câteva posibilități de dezvoltare continuă. Cum ar fi creșterea razei de comunicare între bordul meu cu mașina (printr-o tehnologie WiFi), sau implementarea unei interfețe destinate calculatoarelor de tip desktop.

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

**Gépjármű műszerfalának működő
modellje, autóipari kommunikációs
rendserek felhasználásával**

DIPLOMADOLGOZAT

Témavezető:
Dr. Csernáth Géza,
egyetemi adjunktus

Végzős hallgató:
Bogdán Norbert

2016

Kivonat

Kulcsszavak: műszaki diagnosztika, autóipari kommunikáció, mikróvezérlős rendszer, műszerfal

A műszaki diagnosztika napjainkban elengedhetetlen tevékenysége a gépjárműiparnak. Az autókat használatuk során, jogi szabályok, illetve a fenntartás miatt kötelezően alá kell vennünk méréstechnikai vizsgálatoknak amelyek információt adnak a rendszer műszaki állapotáról annak lényeges megbontása nélkül. Az információkat, a szenzorokon, illetve jeladókon keresztül tudjuk elérni. Az adatokhoz való külső hozzáférést, a jármű diagnosztikai rendszere biztosítja, amely a jármű által támogatott kommunikációs protokollohoz biztosít hidat.

A diplomamunkám téma, ezeket az autóipari kommunikációs protokollokat, illetve diagnosztikai elveket tanulmányozni, valamint ezek demonstrálására, egy olyan mikróvezérlős rendszer összeállítása (mechanikai és elektronikai rendszer), amely egy diagnosztikai csatlakozó áramkör segítségével képes néhány különböző a jármű működésére vonatkozó adatot összegyűjteni, feldolgozni, illetve néhányat a felhasználók számára kijelezni. A rendszerrel egy műszerfálat modelleztem, amely a sebességet, és fordulatszámot mutatókkal, az egyéb adatokat pedig kijelzővel szemléltetik. A dolgozat szerves részét képezi, a rendszert működtető mikróvezérlőre írt szoftver tervezése és implementálása is.

Első lépésként szükséges volt egy hordozható vázat készítsek, amelyre felhelyeztem a műszerfal vázát, illetve a teljes elektronikát. A következő lépés, egy diagnosztikai csatoló áramkör beszerzése volt, amin keresztül elértem a jármű diagnosztikai rendszerét. Ez az áramkör egy ELM327-es, amely lényegében kapcsolatot teremt a jármű diagnosztikai rendszere, és egy soros kommunikációs protokoll között. Ez esetben egy Bluetooth protokoll. A rendszert egy PIC18F4620-as mikróvezérlő irányítja, amely egy HC-05-ös Bluetooth modul segítségével kommunikál a csatoló áramkörrel. A mikróvezérlő kéréseket küld a jármű fele, majd az ezekre kapott válaszokat feldolgozza, és kijelzi a felhasználó fele. A sebesség, illetve fordulatszám mutatók vezérlését léptetőmotorokkal oldottam meg, míg a többi adatot alfanumerikus kijelzőre iratom ki. Továbbá a rendszer részét képezik, a Hall szenzorok amelyek a mutatók alaphelyzetbe állásáért felelnek.

A következő fontos része a dolgozatomnak, a mikróvezérlőre tervezett szoftver implementálása. A szoftvert MPLAB környezetben, C nyelven írtam, és C18-as fordítóval van lekompilálva gépi kódába. A szoftver beírását a mikróvezérlő program memóriájába egy előzőleg beírt, tárrezidens betöltő program segítségével végzem. A szoftver funkcionálitását tekintve, ez végzi a perifériák vezérlését, és a kommunikációs protokoll kódolását, illetve dekódolását.

Miután elkészült a rendszer, teszteléseket végeztem rajta. Először a soros aszinkron kommunikáció helyességét teszteltem, majd a rendszeremet, valós járműhöz kapcsolva, vizsgálva, hogy a kijelzett adatok megegyeznek-e a jármű műszerfalán látottakkal.

A fent leírt rendszert sikerült megvalósítanom, illetve működésbe hoznom, megismerve ezáltal a diagnosztikai- és autókommunikációs elveket. A témahoz további kiegészítéseket lehetne tervezni, mint például a kommunikáció hatótávolságának növelése, vagy a rendszer tesztelését könnyítő, valamint az elvek működését igazoló szoftverek implementálása valamelyen operációs rendszerre.

Abstract

Keywords: technical diagnostics, automotive communications, microcontroller system, dashboard

The technical diagnosis of cars is an indispensable activity in the motor vehicle industry. Cars being in use, due to legal regulations and maintenance, have to go under technical measurements from time to time. This is needed in order to get valuable information about the cars technical condition without any significant intrusion in to the cars system. This type of information can be accessed via the sensors and signal transmitters installed in the car. External access to this data, is ensured by every modern cars diagnostic system. This system provides a bridge to communicate thanks to the communication protocols implemented in it.

The topic of my thesis is to study these car industry specific communication protocols, and diagnostic principles. Also in order to demonstrate these notions in action I aim to build a microcontroller based system, consisting of both mechanical and electrical implementations. This system will be able to connect to a car via a diagnostic socket circuit, after a connection has been established we will collect some vehicle specific data, process them and display them to a user. This system simulates a simple version of the dashboard of a car. This dashboard illustrates the speed and rpm of the engine via clock hand-like indicators, and the rest of the data via an alpha numeric LCD. An integral part of the thesis is to design and implement the software that makes the microcontroller work.

First step was creating the mechanical elements of the system. I built a portable frame upon which I placed the frame of my own dashboard, and on a panel I placed all the electronics. Next step was to obtain a diagnostic interface circuit through which I'd be able to access the vehicles diagnostic system. The chosen circuit is named ELM327, through which one can setup a connection between the cars diagnostic system and a serial communication protocol. In our case this is a Bluetooth protocol. The whole system is controlled by a PIC18F4620 microcontroller, which communicates with the diagnostic interface circuit trough a HC-05 Bluetooth module. The microcontroller sends requests to the vehicle, and after it receives a response processing it displays the data to the user. The display of the speed and engine rpm are displayed by controlling small stepper engines, while the rest of the data is displayed on the alpha numeric LCD. Also in order for the clock hands not to pass under a specific value, I control them with Hall sensors, a small magnet attached to the hand stops it when it reaches the zero value.

The next step was to develop the software that makes the system work. The software was written in MPLAB, in the language C, and I used the C18 compiler to translate my code in to machine code. Getting the compiled code in to the program memory of the microcontroller is achieved through storage-resident loading program. The software solves the control of the peripheral devices and the coding/decoding of the communication protocol.

The system described previously was completely built, and the software was completely implemented. It's a functional system which I managed to test several times. Through these test I understood the diagnostic and communication principles of vehicles. The system can be further developed by increasing the range of the communication, going from Bluetooth to Wi-Fi. Or to develop a desktop application that works on and proves the same principles, but in a friendlier environment on a preferred operating system.

Tartalomjegyzék

Ábrák jegyzéke	15
Táblázatok jegyzéke	17
1. Bevezető	18
1.1. Bevezető	18
1.2. A dolgozat témája	19
2. Szakirodalmi tanulmány	20
2.1. Bevezetés a műszaki diagnosztikába	20
2.1.1. A műszaki diagnosztika alkalmazásának célja	20
2.2. A műszaki dianosztika felosztása	21
2.2.1. Fedélzeti diagnosztika	22
2.2.2. Nem fedélzeti diagnosztika	27
2.3. A műszaki diagnosztikában használt kommunikációs protokollok	27
2.3.1. Központosított szabályzó rendszerek	27
2.3.2. Elosztott szabályzó rendszerek	27
2.3.3. LIN: Local Interconnect Network	28
2.3.4. FlexRay	28
2.3.5. CAN: Controller Area Network	28
2.4. Hivatkozások a fejezethez	31
3. Elméleti megalapozás	32
3.1. ELM-327	32
3.2. Mikróvezérlő	33
3.2.1. A mikróvezérlők általános felépítése	34
3.2.2. UART-soros aszinkron kommunikáció	35
3.2.3. Megszakítás rendszer	36
3.2.4. A mikróvezérlők programozása	37
3.3. Léptetőmotorok	37
3.3.1. Unipoláris léptetőmotor	37
3.4. Bluetooth kommunikáció	38
3.4.1. HC-05-ös Bluetooth modul	38
3.5. Alfanumerikus kijelzők	39
3.6. Hall szenzorok	39

4. A rendszer specifikációi és architektúrája	41
4.1. Célkitűzés	41
4.2. A rendszer architektúrája	41
4.3. Hardver specifikáció	43
4.4. Szoftver specifikáció	43
5. Részletes tervezés, és gyakorlati megvalósítás	44
5.1. Hardvertervezés	44
5.1.1. Mechanikai tervezés	44
5.1.2. Elektronikai tervezés	46
5.2. Szoftvertervezés	52
5.2.1. A megvalósított kommunikációs protokoll	53
5.2.2. A Bluetooth modul konfigurációja	53
5.2.3. A mikróvezérlőre írt szoftver komponenseinek magyarázata	55
6. A rendszer felhasználása	64
7. Üzembe helyezés és kísérleti eredmények	65
7.1. A rendszer tesztelése	65
8. Következtetések	67
8.1. Megvalósítások	67
8.2. Hasonló rendszerekkel való összehasonlítás	68
8.3. További fejlesztési lehetőségek	68
9. Függelék	69
Irodalomjegyzék	70

Ábrák jegyzéke

1.1.	A rendszer megvalósítása	19
2.1.	A műszaki diagnosztika csatlakozása illetve információs szintjei	21
2.2.	A boardon lévő, hibát jelző lámpa	22
2.3.	A diagnosztikai csatlakozó lánckiosztása	24
2.4.	A diagnosztikai csatlakozó egy lehetséges elhelyezkedése	24
2.5.	A hibaüzenet formája	26
2.6.	Szabályzó rendszerek típusai	27
2.7.	A LIN busz	28
2.8.	A CAN rendszer felépítése	29
2.9.	A CAN üzenetek felépítése	30
3.1.	Az ELM327-es csatolóáramkör	32
3.2.	Az ELM327 blokk diagramma	32
3.3.	A Harvard architektúra	34
3.4.	A mikróvezérlők belső felépítése	34
3.5.	Az I/O portok konfigurálásához szükséges regiszterek	35
3.6.	Időzítő áramkör	35
3.7.	Számláló áramkör	35
3.8.	Soros kommunikációs idődiagramma	36
3.9.	Bipoláris- és unipoláris léptetőmotorok	37
3.10.	A HC-05-ös Bluetooth modul	38
3.11.	Alfanumerikus kijelző	39
3.12.	Folyadékkristályok	39
3.13.	A Hall effektus	40
4.1.	A műszerfal tömbvázlata	42
5.1.	A mechanikai rendszer váza	44
5.2.	A teljes rendszer	45
5.3.	A motorokat, és a beosztásokat tartalmazó panel	45
5.4.	Fejlesztőlap, PIC mikróvezérlők számára	46
5.5.	A műszerfal blokkdiagramja	47
5.6.	A PIC18F4620-as mikróvezérlő lánckiosztása	47
5.7.	A HC-05 modul bekötése	48
5.8.	A feszültségosztó ellenállások	48
5.9.	A feszültségosztó képlete	49
5.10.	A Hall szenzorok bekötése	49
5.11.	Az LCD modul bekötése	50

5.12. A léptető motorok bekötése	50
5.13. A nyomógombok bekötése	51
5.14. Az RS232-es modul	51
5.15. A rendszer kapcsolási rajza	52
5.16. A szoftver egysrzüsített folyamatábrája	55
5.17. Az LCD inicializálásának lépései	57
7.1. A rendszer tesztelése	66
7.2. A rendszer tesztelése	66

Táblázatok jegyzéke

1. fejezet

Bevezető

1.1. Bevezető

Napjainkban az egyre növekvő elektronikai integráltság, illetve az áramköri kompaktság igénye a mikrovezérlők területén is nagy áttörést hozott. A mikrovezérlők egyre eredményesebbé, és olcsóbbá váltak, méretük is egyre kisebb lett, ezért könnyen lehet őket különböző kommunikációs, és egyéb rendszerekbe integrálni. Ez a paradigmaváltás a járművek fejlődésének terén is exponenciális növekedést hozott. Egyre kompaktabb és bonyolultabb kocsikat láthatunk az utakon, melyek többsége már nagyszámú elektronikus rendszert tartalmaz. Funkciójukat tekintve, ezek a rendszerek a felhasználó biztonságát valamit kényelmét szolgálják, másrész pedig a jármű különböző részeit vezérlik, illetve környezetvédelmi szabályoknak próbálnak eleget tenni. Mint minden rendszer esetében, az autóipari rendszereknél is, a megfelelő, robusztus működés érdekében, valamint a felhasználó zavartalan utazásának biztosításáért, a jármű elektronikus rendszere nagy sebességgel kell kommunikáljon az autó különböző komponenseivel, továbbá a külvilággal.

Gondoljunk bele, hogy például egy, az ütközést előrejelző szenzor milyen gyorsan kell az információt továbbítsa a központi feldolgozó egység felé, hogy az, a felhasználó számára még biztonságos időn belül feldolgozza az adatot és beavatkozzon a jármű vezérlésébe elkerülve az ütközést. Említhetnénk akár az automata váltót vagy a menet stabilizátort ahol szintén egy gyors és "intelligens" rendszerre van szükség.

Habár a mikrovezérlők jóval alulmaradnak teljesítményben a processzorokkal szemben, azonban a sokrétű integrált perifériának továbbá az I/O egységek könnyű kezelhetőségének köszönhetően sok – a járműben használt – rendszer feladatát ellátják; vagyis minden olyan operációt ahol nem szükséges magas számítási teljesítmény, sokkal inkább egy megbízható, egyszerű áramkör, ott jól használhatóak ezek a mikrovezérlős rendszerek. Nem utolsó szempont az sem, hogy a viszonylag magas árakat – amiket a gyártók kérnek az autóikért – ezek a rendszerek olcsóságuk miatt lefele visznek.

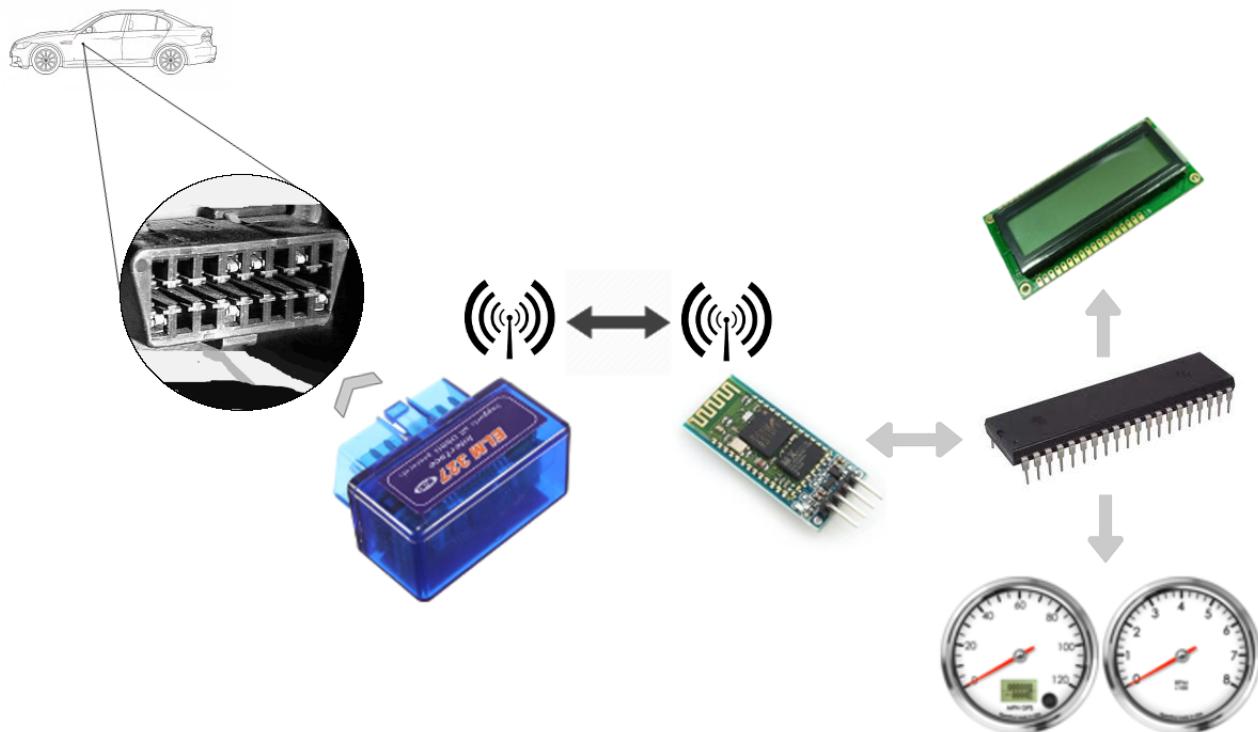
A gyors adatcserén, az adatfeldolgozáson valamint a komponensek külső – illetve belső kommunikációján keresztül jutunk el az autóipari kommunikációhoz. Mint említem, napjainkban egyre több elektronika kerül bele a járművekbe, ezek egyre korszerűbbek és egyre bonyolultabbak. Ma már nem elég a hagyományos rendszerekben használt dedikált adatvonalon történő kommunikáció mivel a bonyolult funkcionálisok, illetve a bonyolultabb vezérlések megvalósítása már drága lenne valamint az összeköttetések száma is véges. Egyre

több olyan rendszer kerül bele a járművekbe, amelyek függnek egymástól, vagyis több vezérlőeszköz együttműködését követelik. Ezért szükséges volt lecserélni a hagyományos pont – pont összeköttetéseket, emellett a lassú sebesség miatt, a soros adatvonalak sem megfelelőek az ilyen rendszerek esetében.

Beláthatjuk, hogy egy olyan buszrendszer szükséges, amely nagy sebességgel képes kommunikálni, ugyanakkor a kommunikációs – illetve egyéb hibákat megfelelően képes kezelni. Napjainkban az autóipari kommunikációs protokollok közül a CAN, a LIN, továbbá a FlexRay protokollok a leggyakrabban használtak az autóiparban. A dolgozatomban ezekről a protokolokról szerezhetünk alapismereteket, továbbá bemutatásra kerül a gépjármű ipar szerves része, a műszaki diagnosztika is.

1.2. A dolgozat témája

A diploma dolgozatom témája, az autóipari kommunikáció – valamint protokollok – tanulmányozása, illetve ennek a demonstrálására egy olyan mikrovezérlős rendszer összeállítása – mikróvezérlős alrendszer és mechanikus alrendszer – amely nagy sebességgel képes, néhány különböző, a jármű működésére vonatkozó adatot összegyűjteni, feldolgozni, és egy részét az emberi felhasználók számára kijelezni. A dolgozat szerves részét képezi, a mikróvezérlő irányítását végző szoftver megtervezése, és implementálása is. A rendszer egy OBD csatolóáramkör segítségével kapcsolódik a jármű kommunikációs rendszeréhez, és megfigyeli a szenzorokat amelyek adatait fel kell majd dolgozza. A rendszer, a járművel Bluetooth-on keresztül kommunikál.



1.1. ábra. A rendszer megvalósítása

2. fejezet

Szakirodalmi tanulmány

2.1. Bevezetés a műszaki diagnosztikába

A diagnosztika fogalma a görög "diagnosis" szóból származik. Megkülönböztető felismerést, valamely hiba elindító okának biztos felismerését jelenti.

A műszaki diagnosztika tehát – a fentiekből kiindulva – műszeres méréstechnikai vizsgálatok összessége, amellyel az adott szerkezet műszaki állapota, annak lényeges megbontása nélkül feltárható vagyis, a műszaki diagnosztika a mechatronikai rendszerek *állapot-felügyeletéhez* szükséges műszaki információt szolgáltatja.

A gépjárműveket, berendezéseket és különböző mechanikai gépeket használatuk során gyakran vetjük alá műszeres - ezen belül diagnosztikai módszerekkel végzett - vizsgálatok alá. Ezeknek a vizsgálatoknak általában két oka van:

1. Fenntartás - karbantartás és javítás - céljából
2. Jogszabály írja elő

2.1.1. A műszaki diagnosztika alkalmazásának célja

Mindkét esetben a vizsgálatok célja elsősorban a biztonságos működés biztosítása a felhasználóra, illetve a környezetére nézve, másfelől pedig a rendszer egészének az állapotminősítése miatt. Vizsgáljuk meg ezeket kicsit pontosabban. A műszaki diagnosztika mint eszköz, az alábbi területeken alkalmazható:

1. A szerkezet műszaki-üzemi állapotának értékelése:
 - A meghibásodás, illetve a meghibásodáshoz vezető okok felismerése
 - A rendszerjellemzők megengedett határértéken belüli megváltozásának ellenőrzése
2. A szerkezet műszaki-üzemi jellemzőinek beállítása, szabályzása hangolása és követő elemzése:
 - Egy rögzítendő paraméter értékének - analóg mérés útján - való beállítása
 - A beállított, vezérelt és szabályzott jellemzők értékének ellenőrzése, egy referencia érték figyelembe vételével

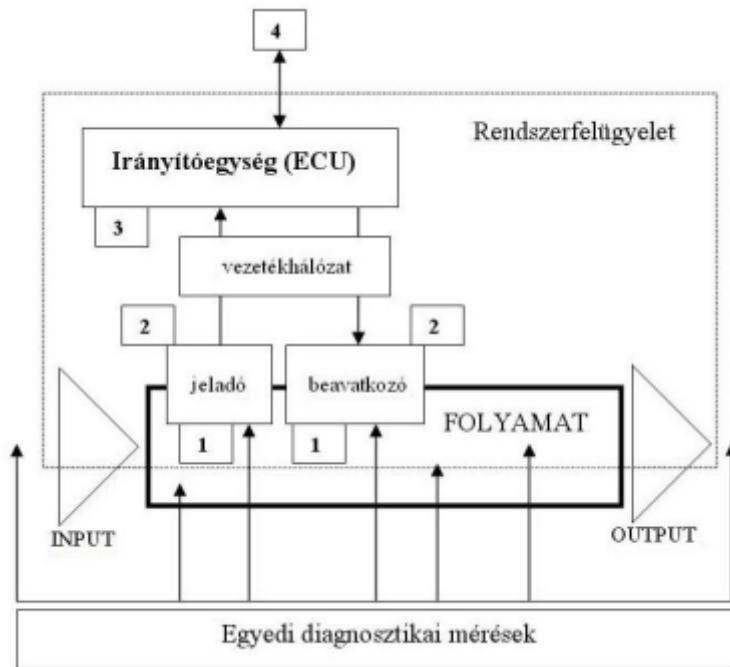
Láthatjuk, hogy a műszaki diagnosztika, járművek esetében számos előnnyel jár, mint például a hibák gyors felderítése, karbantartás hatékonyságának növelése, mérések reprodukciója, valamint a mérési eredmények mentése későbbi felhasználásra illetve fejlesztésre, és még sok más előnyt is meg lehetne említeni.

2.2. A műszaki dianosztika felosztása

A műszaki diagnosztika többféle módon is csoportosítható, de két fő csoportról szoktak beszélni:

- fedélzeti diagnosztika (on-board diagnosztika)
- nem fedélzeti diagnosztika (off-board diagnosztika)

Szükséges megemlítenünk egy többszintűséget is itt. Az információ hozzáférésének helye, továbbá az információ formája és tartalma szerint négy szintről beszélünk. A 2.1 ábrán látható rendszervázlat illetve információs szintek igazak valamennyi, ma használt struktúrájú rendszerre. Ennek megértését, egy belsőégésű motor példáján végezzük el. A bemenet és kimenet közötti folyamat esetünkben magát a motort jelenti. A bemenet (input) egyrészről közvetlen anyagi alkotókból (levegő, tüzelőanyag) és információkból áll, a kimenet is közvetlen anyagi alkotókból (kipufogógáz) és az energiaátalakulásból létrejövő fizikai információkból (hőenergia, hangenergia, stb.) áll. A teljes folyamatnak csupán egy része irányított. A rendszer információs



2.1. ábra. A műszaki diagnosztika csatlakozása illetve információs szintjei

szintjei tehát a 2.1 ábrán 1-től 4-ig terjedő számokkal jelöltek és a következő a jelentésük:

1. szint: itt a fizikai jellemzők közvetlenül jelennek meg (pl. nyomás, hőmérséklet, fordulatszám, stb.). Az első és a második szint közti kommunikációt a jeladók valósítják meg

oly módon, hogy az első szintem lévő, nem villamos fizikai jellemzőket átalakítja villamos jellé. A diagnosztika során, hitelesített diagnosztikai eszközzel kell a fizikai jellemzőket mérni és összevetni őket egy referencia értékkel.

2. szint: A jeladók, illetve a beavatkozók találhatók itt. Ezeknek a paramétereit, és a kimenetükön megjelenő jelet mérjük.
3. szint: A perifériavigszálat szintje. Periféria alatt értjük az irányítóegységhez csatlakozó valamennyi áramkört (annak elemeit, csatlakozóit, vezetékhálózatát). Az irányítóegység lecsatolása után, a központi csatlakozón keresztül elérhető a rendszer valamennyi perifériá-áramköre. A 3. szinten, a rendszer üzeme közben (feszültség alá helyezett rendszer, indítómotorral forgatott motor, járó motor) célszerű kábelezéssel (pl. Y csatlakozó) kell mérni a jeleket.
4. szint: Az irányítórendszerrel történő kommunikáció szintje.

2.2.1. Fedélzeti diagnosztika

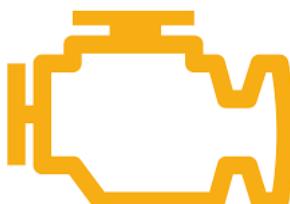
A CARB (California Air Resources Board) levegőtisztaság-védelmi hatósága, felismerve a folyamatos állapotfelügyelet jelentőségét, a gyártók részére előírásban rögzítette a gépjármű-emisszió-korlátozó műszaki rendszereinek fedélzeti ellenőrzési kötelezettségét vagyis a fedélzeti diagnosztikát.

A fedélzeti diagnosztikáról általánosan

A mechatronikai - mechanikai és elektronikai - rendszer állapotát a fedélzeten folyamatosan figyelni képes diagnosztika rendszer esetében beszélünk fedélzeti diagnosztikáról. Ebben az esetben rögtön értesülünk a zavar fellépéséről, illetve a szenzorok állapotáról és valós időben léphetünk közbe.

A fedélzeti állapotvizsgálat a gépjármű rendszerének saját funkciója, vagyis az ehhez szükséges hardver, valamint szoftver a rendszer részét képezik, ténylegesen a szerkezet integrált elemei.

Az első generációs diagnosztikai rendszereket az 1980-as években fejlesztették ki, *OBD I* (On-Board Diagnosis I) néven. Az OBD I alapelvei azok, hogy minden olyan komponenst ellenőrizni kell ami hatással van a kibocsátásra, és a motor vezérlő rendszerrel elektronikus kapcsolatban van, valamint a bekövetkezett hibát tárolni, és a felhasználót figyelmeztetni kell a műszerfalon lévő hibát jelző (MIL-Malfunction Indicator Light-2.2 ábra) lámpával.



2.2. ábra. A boardon lévő, hibát jelző lámpa

Az OBD I-nek van néhány hiányossága, ezek közül a legjelentősebb hiányosság, hogy nem volt elég szabványos a rendszer.

Ennek köszönhetően, a szabványosításra törekedve, és figyelembe véve a SAE (Society of Automobile Engineers) műszaki szabályozásának előírásait, fejlesztették ki az 1990-es évek elején az *OBD II* diagnosztikai rendszert, melynek újdonságai a következők voltak:

- A hibalámpa szabványos jelzési funkciókkal bővült
- A hiba fellépésének mértékét, valamint a környezeti paramétereket is rögzíteni kell a hiba fellépésekor
- Hibatároló olvasása digitális kommunikációs interfészen keresztül, speciális diagnosztikai eszközzel - rendszerteszterrel.
- Szabványos csatlakozók (Data Link Connector: DLC), hibakódok (Diagnostic Trouble Code: DTC)

Az európai szabványosítás ISO-n (International Organization for Standardization) keresztül történt, EOBD (European On-Board Diagnostics) lett a neve.

Felügyeleti módok

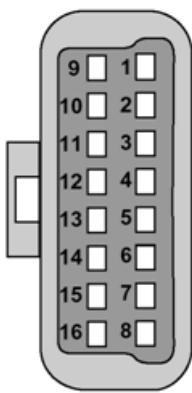
A fedélzeti diagnosztika esetében, mint tudjuk, a rendszer állapota folyamatosan figyelve van, azonban ez a folyamatosság két csoportba sorolható:

1. Állandó folyamatosság, amikor-is a rendszer különböző paraméterei ténylegesen folyamatosan figyelve vannak, és állapotukról azonnali visszajelzést kapunk (például fordulatszámláló, sebességszámláló, hőmérséklet-érzékelő).
2. Periodikus folyamatosságról beszélünk, ha a rendszer paraméterei csak alkalomszerűen, periódusokban vannak ellenőrizve és kijelzve (pl. lambda-szonda). Az alkalomszerű mérés jelensége, jól megfigyelhető, amikor egy - a boardon - hibát jelző égő hatására, annak hibakódját törli a szakember, azonban magát a hibát kiváltó okot nem javítja ki, remélve, hogy az helyrejött magától (például egy koszos szenzor esetében). Ha a hiba még mindig jelen van, a felhasználónak nem egyből jelenik meg újra a hibát jelző fény, hanem csak egy idő múlva, mivel az aktuális szenzor állapota csak időszakosan van megfigyelve.

Az OBD-csatlakozó

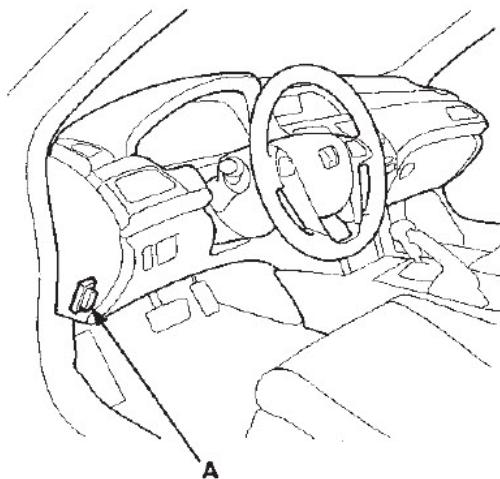
A diagnosztikai csatlakozó aljazatot, az előírásoknak megfelelően, az utastér valamelyik pontjába kell elhelyezni. Napjainkban a járművet gyártó cégek, más-más helyet adtak a csatlakozó aljazatnak, azonban ez általában a következő három helyszín valamelyikében megtalálható: a kormány alatti- vagy melletti részen, az autó közepén lévő kézifék környékén, vagy a hamutartó hátánál. A 2.4 ábrán, az "A"-val jelzett helyen láthatunk egy lehetséges elhelyezkedést.

A diagnosztikai csatlakozó labkiosztása, illetve geometriai méretei a szabványosítás miatt, törvényesről meg vannak határozva. A csatlakozón 16 pólus van, amelyből néhány láb definiált, néhány nem. A nem definiált csatlakozási pontokat a gyártók saját elképzélésük szerint, szabadon felhasználhatják.



2.3. ábra. A diagnosztikai csatlakozó lánckiosztása

A csatlakozó lánckiosztása megmutatja az alkalmazott kommunikációs protokollt, 12 ilyen protokoll van. A szabvány szerint, a diagnosztikai rendszer, automatikusan fel kell ismerje a kommunikációhoz szükséges adatátvitel módját.



2.4. ábra. A diagnosztikai csatlakozó egy lehetséges elhelyezkedése

A 2.3 ábrán látható a diagnosztikai csatlakozó lánckiosztása, a 2.1 táblázat segítségével azonosítjuk.

Mint minden kommunikáció csatorna esetében, itt is az információ átvitel iránya, illetve az ehhez rendelkezésre álló átviteli út szerint az információs kapcsolat lehet:

- szimplex, amikor egyetlen átviteli út áll rendelkezésre és ezen minden csak az egyik irányban van információátvitel
- félduplex kapcsolat, amikor az egyetlen átviteli úton felváltva egyik, illetve másik irányú átvitel zajlik
- duplex kapcsolat, amikor két adatátviteli út áll rendelkezésre és így egy időben minden két irányban lehetséges az információátvitel

Pin	Felhasználás	Funkció
1	Nincs bekötve	-
2	SAE J1850	adatátvitel SAE J 1850 szerint (busz plusz vezeték)
3	OBDII	buszrendszerenél a Vcc csatlakozás
4	SAE J1962	Karósszéria test
5	SAE J1962	Jel test
6	CAN-High (ISO 15765-4 és SAE J2284)	Kommunikációs protokoll
7	K vezeték	Adatátvitel a ISO 9141-2 és ISO 14230-4 szerint a K vezetéken
8	Nincs bekötve	-
9	Nincs bekötve	-
10	SAE J1850	Adatátvitel a SAE J 1850 szerint (busz mínusz vezeték)
11	OBDII	Buszrendszerenél a tesztelés
12	OBDII	Buszrendszerenél az árnyékolás
13	Nincs bekötve	-
14	CAN-Low (ISO 15765-4 és SAEJ2284)	Kommunikációs protokoll
15	L vezeték	Adatátvitel a ISO 9141-2 és ISO 14230-4 szerint az L vezetéken
16	Akkumulátor feszültség	Plusz feszültség

2.1. táblázat. Csatlakozó pinjeinek a funkciói

Tesztelés

A tesztelés, a diagnosztikai eszköznek a fentebb említett helyek valamelyikén lévő csatlakozóra való felhelyezésével kezdődik. Ez az adatkommunikációs kellék a gépjármű irányítóegységeit szabványosított szoftver segítségével éri el. A járművek nem mindenik diagnosztikai módot támogatják. A diagnosztikai módok a következők:

1. mód: a rendszer aktuális adatainak kiolvasása(például fordulatszám, sebesség, hőmérséklet stb.)
2. mód: "Freeze frame" - környezeti paraméterek kiolvasása
3. mód: Diagnosztikai hibakódok lekérdezése
4. mód: Hibakódok, és a tárolt értékek törlése
5. mód: Tesztértékek, és az oxigén-szenzor értékeinek a kijelzése
6. mód: A nem folyamatosan felügyelt rendeltetések értékeinek a kijelzése
7. mód: A még nem állandósult hibák kijelzése
8. mód: Gyártmányspecifikus tesztfunkciók
9. mód: Jármű-információk kiolvasása az irányítóegységből

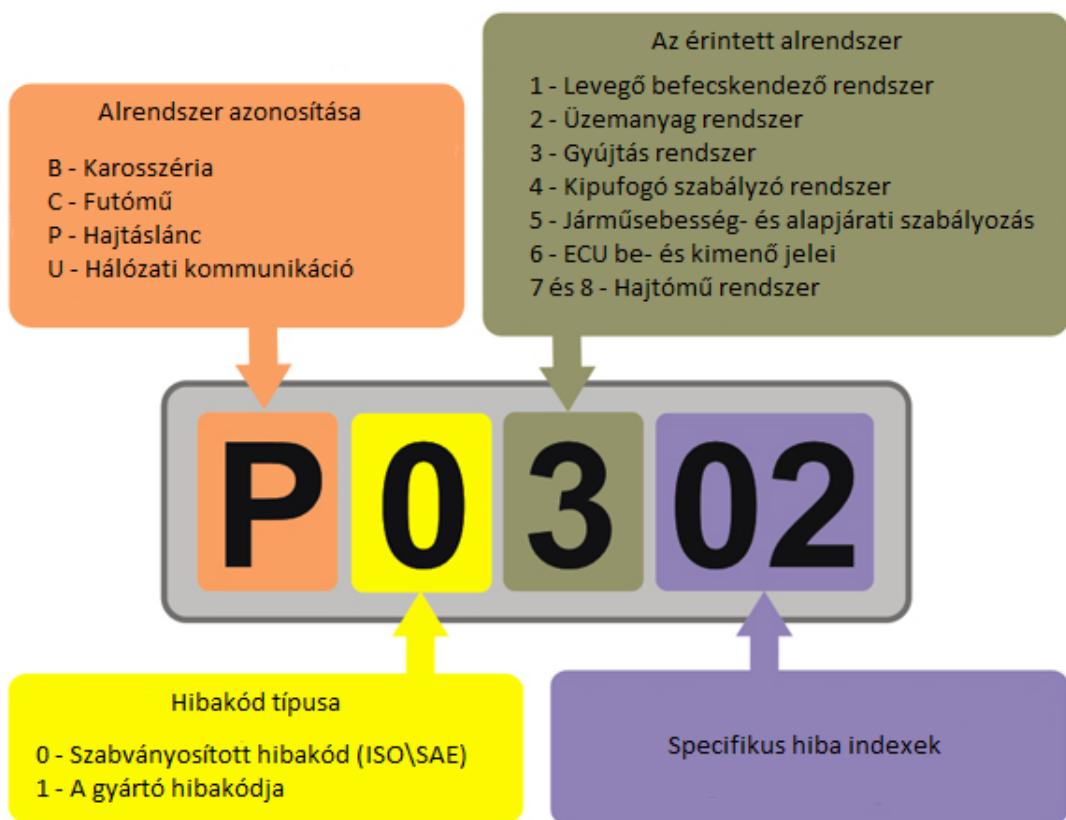
10. mód: Állandósult hibakódok olvasása

Az általam használt diagnosztikai csatoló áramkör esetében, amelyről a következő fejezetben részletebben olvashatunk, a módokat úgy választjuk ki, hogy a kérés első karaktere, a kívánt mód azonosítója (például ha a rendszer aktuális adatait szeretnénk olvasni, akkor a kérés első karaktere a 0x01-es hexadecimális szám lesz).

Hibakódok

A hibakódok vagyis a DTC-k (Diagnostic Trouble Codes) a rendszerben fellépő rendellenességeket tükrözik vissza a felhasználók számára. Ezek a hibák lehetnek időszakosan fellépő, illetve állandósult hibák, de minden esetben a jelenség javítási folyamathoz kötött. Ha valamely hibát kiváltó ok, megszűnik, akkor a kódja törlésre kerül a hibatárolóban a környezeti információkkal (Freeze frame-kel) együtt.

A hibakódok négy információ egységből, és öt karakterből állnak. A 2.5. ábra közepén láthatjuk a hibaüzenet alakját, valamint a négy információ egység jelentését.



2.5. ábra. A hibaüzenet formája

Szükséges itt megemlíteni az úgynevezett. Readiness-kódokat is, amelyek az alrendszerök üzemkészségét jelzik vissza, azért, hogy fel lehessen ismerni, hogy a diagnosztikai felügyelet teljes körű volt-e. Readiness-kódok a diagnosztika végrehajtása alatt generálódnak, és jelzik, hogy a rendszer ellenőrzése el lett végezve, tehát nem jelentenek kontrollt a fellépő hiba felett csupán információ jellegűek.

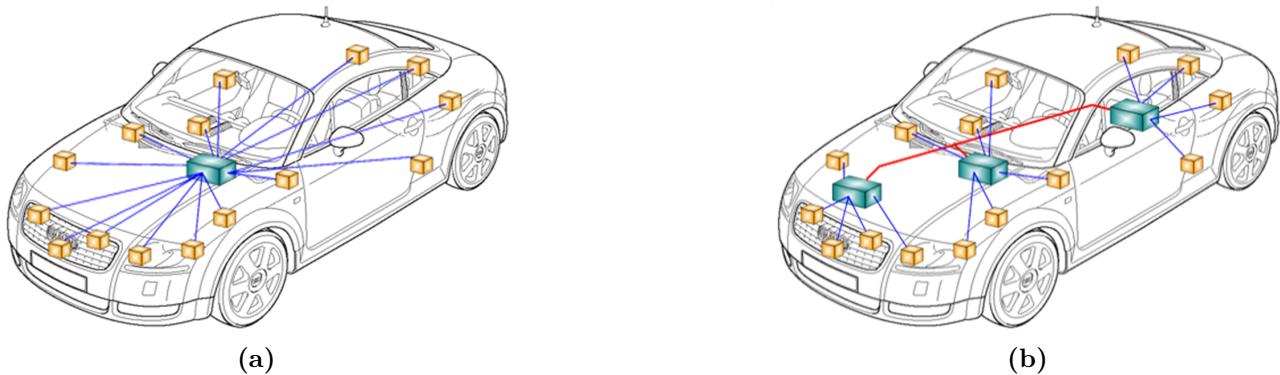
2.2.2. Nem fedélzeti diagnosztika

A nem fedélzeti (off-board) diagnosztika esetében, a vizsgálathoz szükséges hardver illetve szoftver elemek nem képezik a rendszer részét. A mérőeszközököt a vizsgálat időtartamára csatlakoztatni kell (például a gyújtórendszer villamos elemeinek a mérése).

2.3. A műszaki diagnosztikában használt kommunikációs protokollok

A járművekben sok szenzor, és jeladó található, ezért szükséges volt egy rendszer kitalálása amely valamilyen kommunikációs protokollon keresztül összegyűjti az adatokat és ezeket majd továbbítja a felhasználó fele. Napjainkban két típusú szabályzó rendszert használnak:

1. Központosított szabályzó rendszerek (2.6 ábra - a)
2. Elosztott szabályzó rendszerek (2.6 ábra - b)



2.6. ábra. Szabályzó rendszerek típusai

2.3.1. Központosított szabályzó rendszerek

A központosított szabályzó rendszerek esetében egy koordinátor van az architektúrában aki összehangolja a rendszer működését. A koordinátor (master) ciklikusan kéréseket küld a többi eszköz (slave) fele és fogadja azok üzeneteit. A központosított szabályzó rendszer nem hatékony, mivel van hogy a szolgák más-más csatlakozókkal rendelkeznek így nagy számú vezetékezésre lehet szükség ahhoz, hogy egy koordinátorra kapcsoljuk őket, továbbá nehezebb a hibák felderítése, és ha a központi vezérlő egység meghibásodik, az maga után vonja a rendszer szétesését. Ezért van az, hogy a központosított szabályzó rendszereket hamar leváltották az elosztott rendszerek.

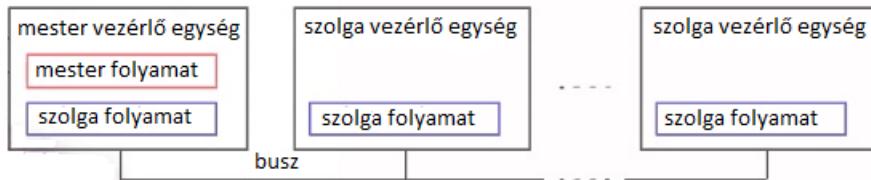
2.3.2. Elosztott szabályzó rendszerek

Olyan soros, adatkommunikációs rendszerek amelyek a tereptartományban való kommunikációra hivatottak. Tereptartomány a térben közeli elemeket jelenti (2.6 ábra - b). Az ilyen rendszerekben, az elemek egyetlen kommunikáció vonalra (buszra) vannak kötve, és önállóan kommunikálhatnak egymással. A rendszer előnye, hogy könnyen bővíthető, olcsó és megbízható a kevés huzalnak köszönhetően.

Az elosztott rendszerek elemei különböző csoportokba sorolhatóak (például kényelmi- vagy komfort elektronika egy csoport, motorvezérlő egy másik), és szerint hogy az adott csoport milyen funkciókat lát el, más-más kommunikációs protokoll dominálhatnak az adott területen. A továbbiakban ezekről a protokollokról lesz szó.

2.3.3. LIN: Local Interconnect Network

A LIN egy soros kommunikációs protokoll amely mester-szolga kialakításon alapszik. A hálózaton egy mester- és egy, vagy több szolga csomópont lehet, úgy, hogy a csomópontok száma ne haladja meg a 16-ot. A mester csomópont egy mester- és egy szolga folyamattal rendelkezik, míg a szolgák csak egy szolga folyamattal (2.7 ábra). A mester folyamat látja el az üzenetek időzítését, fejlécek küldését és a válaszok fogadását. A szolga folyamat dolga pedig a fejlécek fogadása, értelmezése és szinkronizációja, valamint a válaszrészletek küldése vagy fogadása. A LIN protokoll az OSI rétegekből az adatkapcsolati- illetve a fizikai réteget valósítja meg.



2.7. ábra. A LIN busz

Az üzenetküldést mindenkor a mester folyamat kezdi, és erre pontosan egy szolga aktiválódik. Az üzenet azonosítója mindenkor az üzenet tartalma és nem a célállomás t jelzi. Lehetséges az üzenetszórás, valamint a szolgák egymást közt is kommunikálhatnak mester beiktatása nélkül.

Az autóipari kommunikációs protokollok közül ennek a protokollnak a kiépítése a legolcsóbb, viszont egy flexibilis, megbízható, kiszámítható rendszerről beszélünk. A LIN protokollt általában a kényelmi- és a komfort területeken lévő rendszereknél használják (például ablakemelő, központi zár, fűtésrendszer, stb.).

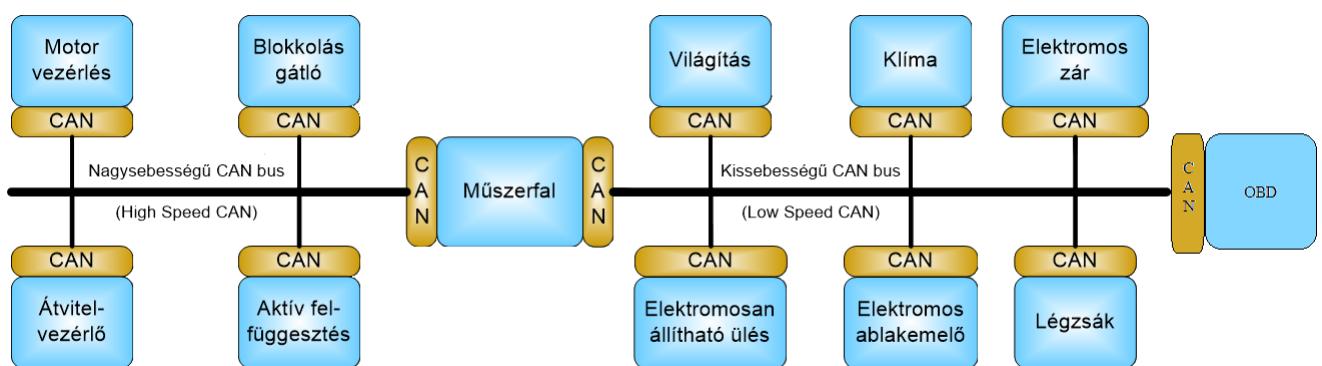
2.3.4. FlexRay

A FlexRay egy szinkron, és aszinkron átvitelt is támogató, vezetékes protokoll. Viszonylag gyors, és nagy sávszélességgel rendelkezik, ezért előszeretettel használják olyan rendszereknél, ahol a hagyományos mechanizmust, elektromos vezérlőkkel akarják helyettesíteni. (például a hagyományos, mechanikus kormányvezérlés felcserélése szervó vezérléssel, továbbá a hidraulikus fékkerekek elektromos vezérlése, stb.).

2.3.5. CAN: Controller Area Network

A CAN protokollt a Bosch cég fejlesztette ki, amit az ISO szabványosított. Ez a protokoll napjainkban már elkerülhetetlen a járművekben, az egyre növekvő vezérlőeszközök száma, valamint a vezérlőeszközök egymástól való függése miatt, mivel a protokoll támogatja az elosztott működést (2.3.2 alfejezet).

A CAN protokoll egy olyan rendszer hálózatot segít megvalósítani, amelyben a vezérlőszkök egy kétvezetékes, soros buszon keresztül, üzenetkeretekkel kommunikálhatnak egymással. Mivel a CAN protokoll olcsó, megbízható, és elektromágneses zajjal szennyezett környezetben is működik, általában valós idejű rendszereknél használják. Attól függően, hogy egy adott valós idejű alrendszer funkciója mennyire fontos, illetve milyen sebességgel kell adatokat továbbítson, a hálózat sebessége 200 kbps-tól 1 Mbps-ig terjedhet, vagyis néhány rendszer kis-, míg mások nagy sebességgel kommunikálnak (2.8 ábra). A jármű diagnosztikai rendszere (OBD) is erre a CAN buszra van rákapcsolva, így külső hozzáférésünk van a csomópontok információihoz. A CAN protokoll az ISO/OSI modell adatkapcsolati és fizikai rétegét implementálja, amely bővíthető magasabb szintű protokollokkal amelyeket az alkalmazási réteg ír le. A CAN hálózat nagyon könnyen bővíthető új csomópontokkal, anélkül hogy más állomások szoftverét, vagy hardverét változtatni kéne.



2.8. ábra. A CAN rendszer felépítése

A CAN fizikai rétege

A fizikai réteg végzi a csomópontok közti tényleges jeltovábbítást. Felelős a digitális jelek analóggá- és vissza való alakításáért, a bitidőzítésért, és a szinkronizálásért.

A CAN adatkapcsolati rétege

Az adatkapcsolati rétegnek alapvetően két feladata van. Elsősorban a busz felől jövő üzenetek szűrése, vagyis mely üzenetet szükséges megtartani és melyiket kell elvetni. Másodsorban ez a réteg végzi a megfelelő üzenetkeretek készítését, vezéri a döntéseket, illetve felismeri és jelzi a hibákat.

A CAN hálózati üzenet váltása

A CAN úgynevezett multi-mester protokoll mivel nincs egy hagyományos értelemben vett vezérlő csomópont, minden egység egyenrangú. Tehát minden üzenet prioritásától függően lesz egy csomópont adó, vagy vevő. Egy időben csak egy adó lehet, ilyenkor minden más csomópont vevő lesz.

Az adatátvitelkor nincs címzettje az üzenetnek, egy azonosítója van, amely a tartalom mellett az üzenet prioritását is definiálja, ez a busz elfoglalásában vállal szerepet. minden állomásnak van egy vezérlő egysége, amelynek az a feladata, hogy az adatokat és az azonosítókat

a CAN chipnek továbbítja. A CAN chip összerakja az üzenetet, és miután megszerezte a buszhozzáférési jogát, kiküldi azt. minden egység, amely helyesen vette az üzenetet, ellenőrzi, hogy az üzenet neki szól-e, ha nem eldobja különben feldolgozza azt. Az üzenetek felépítése a 2.9 ábrán látható. A CAN protokoll két keretformát támogat amelyek csak az azonosító (ID) hosszában különböznek.



2.9. ábra. A CAN üzenetek felépítése

A nem destruktív bitenkénti döntés

Mivel a CAN valós idejű rendszerekkel dolgozik, nem elég csak a gyors sebesség biztosítása, hanem egy gyors buszallokációs mechanizmus is szükséges ha több állomás szeretne üzenetet váltani. A valós idejű adatok feldolgozásának gyorsasága különböző lehet. Például egy gyorsan változó mennyiséget - mint a fordulatszám, motor-fogyasztás, sebesség, stb. - sokkal gyorsabban kell átvinni, vagyis ezek az üzenetek nagyobb priorisással kell rendelkezzenek.

Az üzenetek prioritási szintjei bináris számokban vannak kódolva, tehát a buszhozzáférési problémák bitenkénti döntéssel oldhatók meg. Az "ÉS" kapcsolatra hivatkozva, tudjuk hogy a logikai 0-ás felülírja a logikai 1-est. Így van ez a buszhozzáférés alkalmával is, a domináns állítás felülírja a recesszívet, vagyis az adó szerepet azok a csomópontot veszítik el amelyek átvitele recesszív (logikai 1-es), és megfigyelése pedig domináns (logikai 0-ás). Az összes vesztes csomópont az üzenet vevőjévé válik, és nem próbálkozik üzenetküldéssel, mindaddig amíg a busz fel nem szabadul.

Hibadetektálás

A CAN protokoll előnyeihez sorolhatóak a hibadetektáló mechanizmusai. A CAN nem nyugtázó üzeneteket küld, hanem konkrétan jelzi az előforduló hibákat. A következő három mechanizmus implementálja az üzenetek szintjén:

- Cyclic Redundancy Check (CRC)

A CRC a keretben levő információt biztosítja redundáns ellenőrző bitek hozzáadásával az átvitel végén. A vevő ezeket a biteket újra kiszámolja a fogadott adatok alapján, és ellenőrzi, hogy megegyeznek-e. Ha nem egyeznek meg, CRC hiba lépett fel.

- Keret ellenőrzés

A keret ellenőrzés esetében, tudva a mezők fix formátumát, az mechanizmus az átvitt keret struktúráját ellenőrzi. Ha nem talál, formátum hiba lépett fel.

- ACK hibák

A keretet fogadó résztvevők minden esetben pozitív nyugtával konstatálják a kapott üzenetet. Ha az adó nem kap visszajelzést, ez azt jelentheti, hogy az átvitel során olyan hiba lépett fel, amelyet csak a vevők érzékeltek, az ACK mező meghibásodott, vagy nincsenek vevők. Az ilyen típusú hibákat ACK hibáknak nevezik.

- Monitorozás

Minden adó fél, egyben monitorozza a buszt is, így fel tudja ismerni, az adott, illetve vett adatok közti eltéréseket.

- Bitbeszúrás

A bitek kódolása a NRZ (non-return to zero) mechanizmussal van megoldva, vagyis minden az a feszültség van a vonalon amit az ábrázolt bit határoz meg. Két 1-es (high) állapot között, nem tér vissza nullára, megszakítás nélkül 1-ben marad. A szinkronizációs élek bitbeszúrással generálódnak, ami azt jelenti, hogy 5 azonos bit után az adó 6.-nak egy komplementer bitet szűr be, ezt a vevő eltávolítja. Az ellenőrzés a beszúrási szabálynak megfelelően történik.

A hálózat konzisztens működését, tehát a fenti mechanizmusok biztosítják. Egy hiba fellépés esetében a csomópont megszakítja a küldést, így a vevők nem kapnak hibás üzenetet, majd automatikusan újraküldi azt.

A CAN protokoll adatainak megbízhatósága igen magas, így bőven megfelel a valós idejű rendszerek követelményeinek. A jellemző eseményvezérltség miatt (a kommunikáció egy esemény bekövetkezése miatt kezdődik el), egy idő hatékony autóipari kommunikációs protokollról beszélhetünk, a kommunikációs időt tekintve, mivel nincsenek elpocsékolta időszeletek az információ hiánya miatt.

2.4. Hivatkozások a fejezethez

A fejezet a következő anyagokban olvasott gondolatok alapján íródott: [1], [2], [3], [4], [5], [6], [7], [8]!

3. fejezet

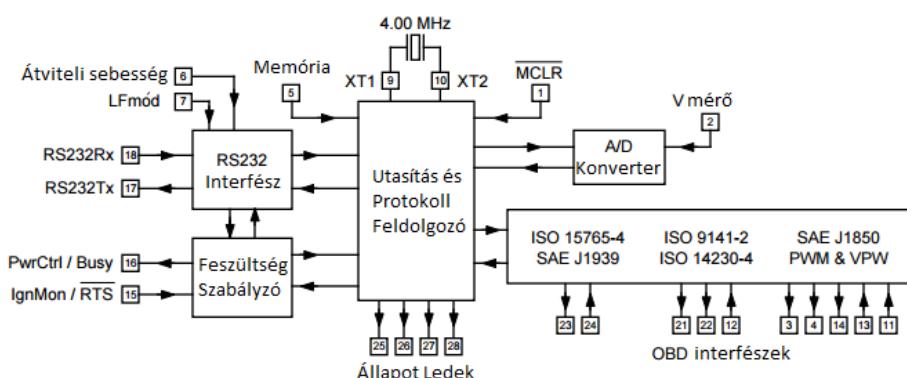
Elméleti megalapozás

3.1. ELM-327

Az előző fejezetben tárgyaltuk, hogy majdnem minden ma gyártott autónak a törvényi előírásoknak megfelelően rendelkeznie kell egy interfésszel amelyen keresztül csatlakozhatunk a diagnosztikai berendezéséhez. Egy ilyen csatolóáramkör az ELM327 -is (3.1 ábra), amely mint összekapcsolja az OBD portokat, egy soros kommunikációs interfésszel, amelyen keresztül adatokat küldhetünk, illetve fogadhatunk egy általunk használt feldolgozó egységgel.



3.1. ábra. Az ELM327-es csatolóáramkör



3.2. ábra. Az ELM327 blokk diagrammja

Az ELM327 öt fontos modulból áll (3.2 ábra):

1. RS232 interfész: Ezen keresztül kommunikál a az ELM a külvilággal. Ez a modul szabályozza az átviteli sebességet (6-os pin), az üzenet végére beszúrandó nyugtázó karaktereket (7-es pin), valamint fogadja- (18-as pin) és kiküldi (17-es pin) az adatokat.
2. Feszültség szabályzó: Ez a módul felelős a rendszer energia ellátásáért, illetve szabályozza a működési módokat a jármű gyújtásától függően (15-16-os pinek).
3. Utasítás és protokoll feldolgozó: Lényegében ez az ELM327 vezérlő egysége. Ehhez kapcsolódik a másik 4 modul, illetve egy órajel generáló oszcillátor amely a működési órajelet biztosítja. Itt történik az autóból érkező adatok feldolgozásra, dekódolása, illetve a jármű által támogatott protokoll kiválasztása.
4. A/D konverter: A jármű szenzoraitól érkező analóg jelet, digitális jellé alakítja.
5. OBD interfések: a megfelelő pinek feszültség szintjei szerinti protokoll elosztás történik itt

Mint említettem az ELM327 csatolóáramkör sorok kommunikációs interfészen kommunikál a felhasználóval, esetünkben egy Bluetooth modulon keresztül. Az ELM327 paramétereit, könnyen be lehet állítani "AT" parancsok segítségével, illetve az üzenetek hexadecimális számokkal vannak kódolva. A rendszer működés közbeni visszajelzését, 4 darab LED segíti, amelyek az RX, és TX csatornákra vannak kötve, illetve a betáplálást jelzik.

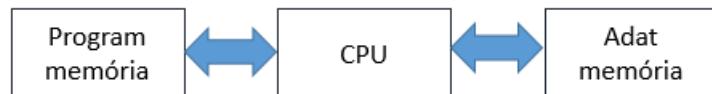
Az ELM327 számos különböző OBD protokolloval támogat, ezek két nagy csoportba sorolhatók: SAE J1850/J1939, és ISO 9141-2/14230-4/15765-4. A protokollok a kommunikációs sebességen, az kommunikációs feszültség szintekben, illetve az üzenetek hosszában különböznek. A küldött kérést az ELM értelmezi, feldolgozza, és a jármű által támogatott kommunikációs protokollnak megfelelő alakba formálva elküldi azt a diagnosztikai rendszernek, amely felől kapott választ szintén a soros kommunikációs protokollnak megfelelő alakban továbbítja, a kérést indító rendszer fele.

Az üzenetek felépítése minden esetben a diagnosztikai mód (2.2.1 alfejezet) hexadecimális számban való megadásával történik. Vegyük egy példát: a jármű sebességét szeretnénk lekérni. Ez a paramétert, a rendszer aktuális adatainak lekérésével nyerhetjük ki, vagyis az 1-es mód. A kérés tehát így kell kinézzen: 01 0D. Vagyis a 01 a szükséges diagnosztikai mód, a 0D pedig a paraméter ID kódja (PID-je), vagyis a sebesség ID-ja. A válasz első bájta egy nyugtázó szám lesz, vagyik a kérés első bájtjához hozzáadva a 40-es hexadecimális, a második bájt pedig a PID szám lesz, vagyis melyik kérésre érkezett válasz. Ezt követően jönnek a hasznos adatbritek. [9]

3.2. Mikróvezérlő

A mikróvezérő egy kisméretű, egyetlen lapkára integrált számítógép, amely utasítások elvégzésére képes, és általában vezérlési feladatak kielégítésére használják. A szilícium lapka amelyre integrálva van a kis rendszer, a vezérlőegységen kívül magába foglalja az adat- és programme-móriát, illetve manapság már viszonylag sok perifériát is megtalálunk benne (Timereket, A/D-D/A átalakítókat, RS232, SPI, I2C, CAN kommunikációs buszokat, PWM modulátort, USB, stb). A mikróvezérlőkre általában Harvard, illetve módosított Harvard architektúra jellemző.

Az említett architektúrák lényege, hogy fizikailag is különálló memóriákat tartalmaznak, amelyek lehetővé teszik, hogy a központi feldolgozó egység ugyanabban az időpillanatban tudjon műveletet- és adatot is olvasni, illetve írni.



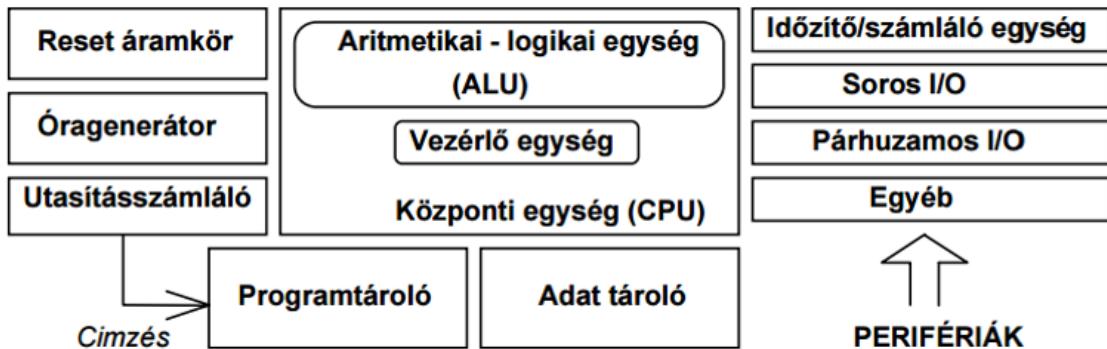
3.3. ábra. A Harvard architektúra

A módosított Harvard architektúra általában két pontban térhet el a Harvardtól:

1. Külön adat- és utasítás cache található benne, miközben minden két memória címteret illetve memóriát használja.
2. Adatutat nyit a programmemória és a CPU között, lehetővé téve ezzel a programmemória szavainak csak olvasható adatként való kezelését.

3.2.1. A mikróvezérlők általános felépítése

A mikróvezérlők rendelkeznek tehát egy központi feldolgozó egységgel, amely egy vezérlő egységből, és egy aritmetikai- logikai egységből (ALU) áll. Emellett minden mikróvezérlőben található egy reset áramkör, amely minden áramkört, illetve regisztert egy jól meghatározott kezdő állapotba hoz, továbbá van egy órajel generátor, amely a rendszer működéséhez szükséges órajelet biztosítja. Találunk egy utasítás számláló regisztert is, amely a következő utasítás címét tárolja, illetve egy program- és adat tároló memóriát (3.4 ábra).

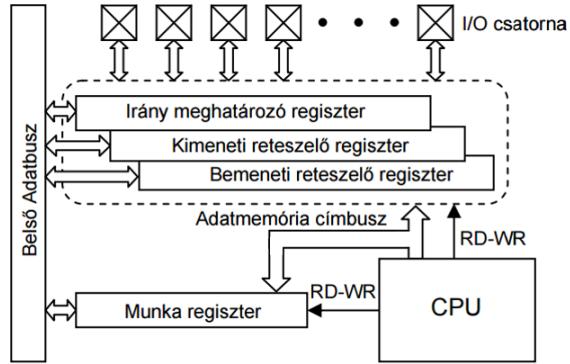


3.4. ábra. A mikróvezérlők belső felépítése

Ami változik vezérlőről-vezérlőre, az a hozzájuk csatolt perifériák. Típusról függően a következő perifériákat tartalmazhatják:

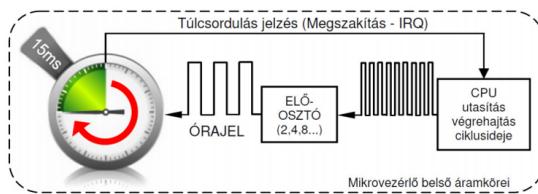
1. Az alap perifériák:

- I/O portok: ezek segítségével tudunk kommunikálni a külvilággal, adatokat tudunk kinyerni- illetve beolvasni az I/O portokon keresztül, a port megfelelő konfigurálásával.



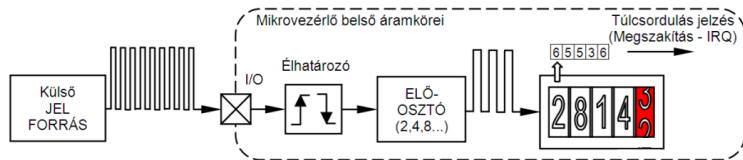
3.5. ábra. Az I/O portok konfigurálásához szükséges regiszterek

- Időzítő áramkör: egy időintervallum pontos kimérésére használjuk.



3.6. ábra. Időzítő áramkör

- Számláló áramkör: impulzusok megszámlálására használjuk.



3.7. ábra. Számláló áramkör

2. Kiegészítő perifériák:

- Compare: meghatározott időpillanatban való digitális jelek előállítására használjuk.
- Capture: periódusmérésre használjuk, egy jel két változása közötti (felfutó és lefutó él) időbeni különbség meghatározása.
- Impulzus szélesség modulátor (PWM): állandó periódusidejű jelek kitöltési tényező-jének változtatására használjuk.
- A/D-D/A átalakítók: analóg elektromos mennyiséget digitalizál, illetve digitális mennyiséget alakít analóggá.
- Szinkron és aszinkron soros adatátviteli interfések.

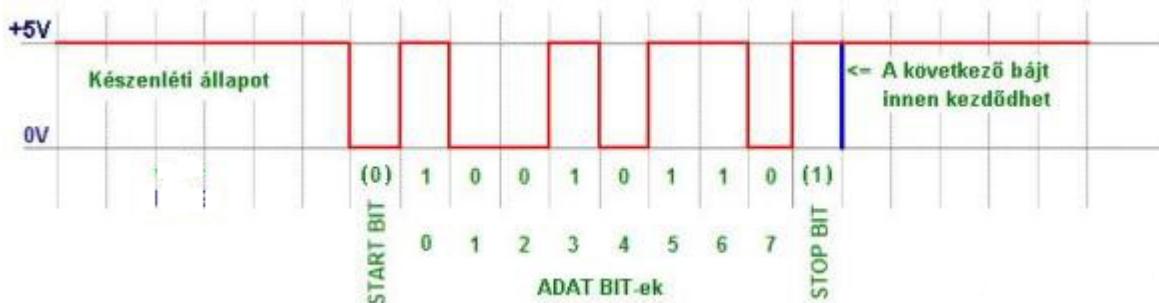
3.2.2. UART-soros aszinkron kommunikáció

A soros kommunikációs modul igen fontos része a napjainkban használt mikróvezérlőknek. Manapság nagyon elterjed jelenség az, hogy minden rendszernek van egy soros kommunikációs

interfésze azért, hogy kommunikálni tudjon a külvilággal, pontosabban más rendszerekkel (PC, mobil, stb). A mikróvezérlők tehát egy beépített UART modullal végzik el ezt a kommunikációt. A modul két lábon (pinen) keresztül kommunikál a külvilággal. Van egy RX lab az adatok fogadására, illetve egy TX lab az adatok kiküldésére.

A kommunikáció kezdetén, a felek egy közös kommunikációs sebességre (baud rate) kell ráálljanak. A küldő fél tehát egy meghatározott frekvenciával kell küldje az adatokat, a vevő fél pedig ugyanazzal a frekvenciával kell fogadja őket. Ez az aszinkron kommunikáció elengedhetetlen része, mivel itt a kommunikáló feleknek nincs egy közös órajel (szinkron kommunikáció) amelyet mindenkor követhetnék. Az adat, minden esetben bitenként lesz továbbítva. Mind a küldő, mind a fogadó modul rendelkezik egy általában 8 bites regiszterrel, amelybe megszakításokkal jelezve, bitenként jönnek be az adatok. Miután megtelt a regiszter, az adatokat kiolvashatjuk, vagy kiküldhetjük.

A küldés kezdetekor az adatvonal a logikai 1-es szinten van, ezután kezdődik el a kiüldés. Először a start bit küldődik ki, amely vagyis egy logikai 0. Ez után kezdődik az adat bitenkénti kiküldése. Miután elküldődik a 8 bit, opcionálisan következhet egy paritás bit. Paritás bit a hibajavításhoz szükséges, ezt előzőleg be kell állítani a kommunikációs modulnál. Végezetűl következik egy stop bit, amely a küldés végét jelzi egy logikai 1-es szinttel.



3.8. ábra. Soros kommunikációs idődiagramma

Fogadás esetén, miután a vevő megbizonyosodott, hogy egy start bit érkezett, utána minden bitidőben vesz egy mintát, megtöltve ezzel a adatok fogadására szánt regisztert.

3.2.3. Megszakítás rendszer

A megszakítás rendszer egyik talán legelőnyösebb funkciója a mikróvezérlőnek. Segítségével, a központi feldolgozó egység több perifériától is tud adatot fogadni anélkül, hogy a program futását zavarná, illetve hosszabb ideig tétlen állapotba kerülne.

Egy beérkezett adat esetén, az ezt feldolgozó periféria egy megszakítás jelzést küld a központi feldolgozó egységnek. A CPU azonnal elmenti a program regisztereit, és aktuális paramétereit, majd kiszolgálja a megszakítást. Ezután kiolvasva az elmentett adatokat, visszatér a program végrehajtásához, pont oda ahol abbahagyta azt. A mikróvezérlők általában vektorizált megszakításokat használnak, vagyis egy vektorban vannak tárolva a megszakítást kiszolgáló ru-

tin kezdőcímére ugró utasítások. Több ilyen megszakítás vektor esetében, prioritási sorrendek vannak, amelyet a hardver értékel ki.

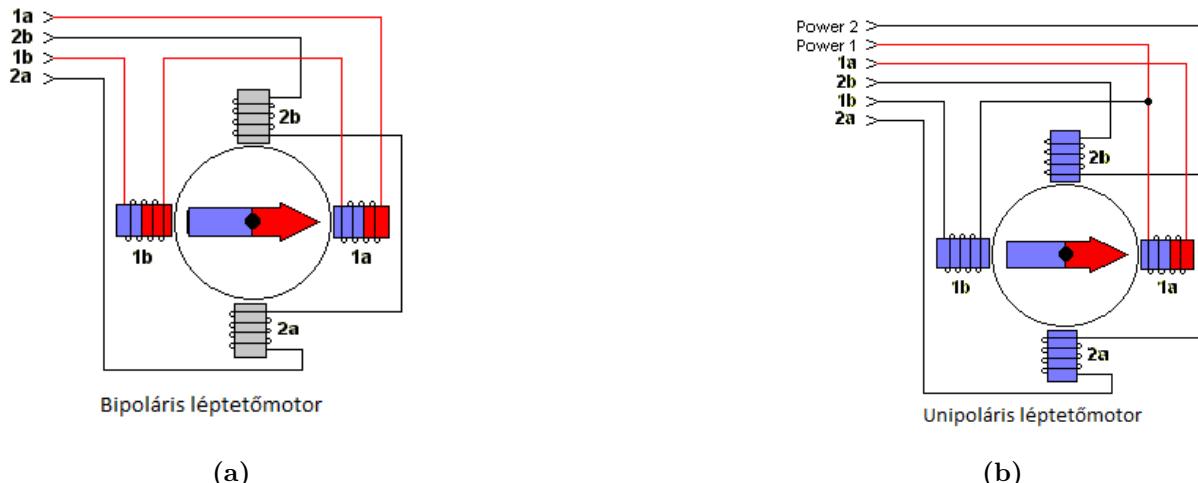
3.2.4. A mikróvezérlők programozása

Alapvetően két módszer van a mikróvezérlők programmemóriájának feltöltésére. Az első módszer, az erre specializálódott áramkörrel való, közvetlen EEPROM vagy FLASH beírás, a második módszer pedig egy tárrezidens betöltő program használata. A második módszerhez egy olyan architektúra szükséges amely megengedi a saját programmemória törlését, írását, illetve olvasását (például a módosított Harvard architektúra).[10], [11]

3.3. Léptetőmotorok

A léptetőmotor egy elektromos árammal működtetett, digitális jelekkel vezérelt beavatkozó elem, melyet általában pozíció szabályzás esetén használnak. Mivel a léptetőmotorokban nincs szénkefe, ezért megbízható, viszonylag sokáig lehet használni. Általában akkor használjuk őket amikor valamely precíz, előre meghatározott pozíció digitális jelformában van megadva, és ezzel a digitális jellel vezérelve szeretnénk valamit eljuttatni az adott pozícióba.

Minden típusú léptetőmotor esetében a tekercselés az álló részben van, valamint a forgó részbe nincs árambevezetés. A forgó rész általában lágy vas, vagy állandó mágnes. A léptetőmotoroknak két fajtája létezik, az unipoláris és a bipoláris. A bipoláris motorok vezérlése bonyolultabb, viszont a motor nagyobb erő kifejtére képes.



3.9. ábra. Bipoláris- és unipoláris léptetőmotorok

3.3.1. Unipoláris léptetőmotor

Az unipoláris léptetőmotrokból két tekercs található (3.9-b ábra) és még a bipoláris motoroknak 4, addig az unipolárisnak 5, 6 vagy annál több kivezetése van (a középleágazások határozzák meg). Általában a középleágazásokat a tápegységre kötik, míg a többi pontot, a digitális vezérlő jelet adó csatlakozóra. Általában a léptetőmotorok meghajtásához, egy köz-

tes teljesítmény elektronikai áramkört használnak a megfelelő teljesítmény, és fordulatszám eléréséhez. Ilyen áramkör például az ULN2003A [12].

ULN2003a

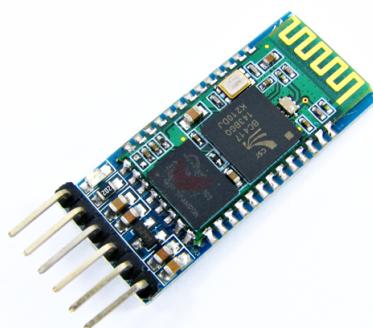
Az ULN2003A egy NPN típusú, Darlington tranzisztorokból álló tömb. Mivel a Darlington tranzisztorokkal viszonylag nagy áram- és feszültség erősítést lehet elérni, ezért használják léptetőmotorok meghajtásához, mert általában a léptetőmotort vezérlő egység (mikróvezérlő), nem képes a motor konzisztens működéséhez szükséges áramot- illetve feszültséget biztosítani [13].

3.4. Bluetooth kommunikáció

A Bluetooth adatcseréhez használt, rövid hatótávolságú, vezeték nélküli szabvány. Alkalmazásával beágyazott rendszerek között létesíthetünk egy rövid hatótávolságú, rádiós kapcsolatot. Kis energia fogyasztása, és a viszonylag nagy sebessége miatt előszeretettel használják hordozható rendszerek esetében. Sok fajta Bluetooth modul létezik a piacon, egy ilyen a HC-05 modul is.

3.4.1. HC-05-ös Bluetooth modul

A HC-05-ös Bluetooth modul, egy kis méretű, a mikró elektronikában elterjedt, Bluetooth kommunikációt megvalósítani képes, soros kommunikációs áramkör (3.10 ábra). A modul két munkaüzemmódban- illetve három munkaszerepben képes dolgozni. A két munkaüzemmód a fogadás-küldés, a másik meg az automatikus csatlakozás. A szerepeket tekintve pedig, mester, szolga, valamint visszacsatolási szerepekben képes működni. A modul konfigurálására rengeteg "AT" parancs áll rendelkezésre. Az előbb említett munkamódokat is ezekkel kell beállítani. Mivel a modul viszonylag nagy sebességgel képes kommunikálni, és megbízható, ezért jól használható kisebb teljesítményű rendszerek esetében [14].



3.10. ábra. A HC-05-ös Bluetooth modul

3.5. Alfanumerikus kijelzők

Az alfanumerikus kijelző egy olyan folyadékkristályos kijelző típus, amelyet számjegyek- illetve betűk egy jobban felismerhető, igényesebb formában való megjelenítésre használják. Alapja egy elektromos áram vezetésére alkalmas kristályos anyag.



3.11. ábra. Alfanumerikus kijelző

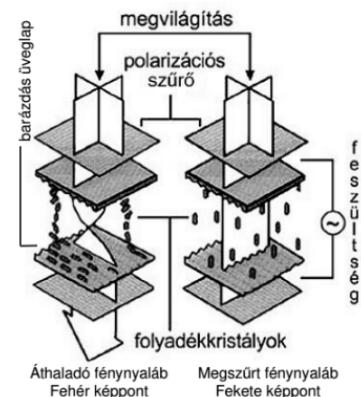
fény továbbhaladva a második szűrőbe ütközik, azon nem megy át, vagyis az elektróda által lefedett sík elfeketedik (3.12 ábra).

Az alfanumerikus kijelző (LCD) a karaktereket sorokba- illetve oszlopokba jeleníti meg, egy előre definiált, memoriában tárolt, általában 5x8-as karakterszettet felhasználva. minden LCD-ben van egy LCD meghajtó, ami kommunikál a vezérlő egységgel (mikróvezérlővel). A kijelző működéséhez előre definiált függvények segítségével inicializálni kell azt, azaz alaphelyzetbe szükséges hozni a megfelelő működésért [11].

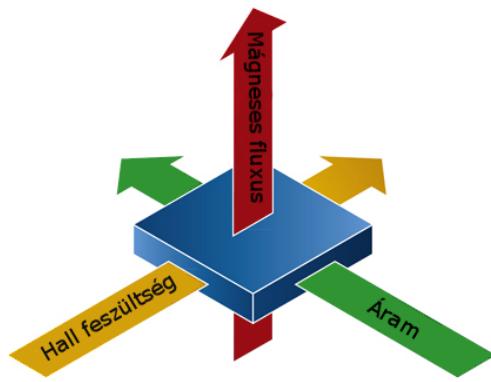
3.6. Hall szenzorok

A Hall szenzorok működése a Hall fizikai hatáson alapul, amely lényege, hogy ha egy félvezető lapkán áram halad át, és ezt a lapkát egy rá merőleges mágneses térbe helyezzük, akkor az áram folyására merőleges irányban a lapkán, potenciál különbség miatt feszültség keletkezik. A Hall szenzorokat általában pozíció, elmozdulás és sebesség érzékelésére használják.

A folyadékkristályok tulajdonsága, hogy nyugalmi állapotban, rendezetten, hosszú egyenes alakjukból kifolyólag, hosszanti tengelyükkel párhuzamosan helyezkednek el. Általában ez a párhuzamosság valamely, a kristályokat közrezáró lapok szerint történik. Ha a kristály-réteget egy vékony réteggel közrezárnak, és ezt a kis tartály kiegészítjük lineáris polarizációs szűrőkkel amelyek egymással derékszöget zárnak be, akkor elérjük, hogy a folyadékkristályok az elektromosság hatására megváltoztassák helyzetüket, vagyis átengedjék a fény, vagy épenséggel elzárják azt. Ha feszültséget kapcsolunk a kijelző elektródájára, akkor a molekulák úgy rendeződnek, hogy a polarizációs síkot nem forgatják el, tehát a



3.12. ábra. Folyadékkristályok



3.13. ábra. A Hall effektus

A Hall szenzorokat széles körben használják az autóiparban. Kezdve a sebesség- és fordulatszám méréstől, a pedálok szögének érzékelésén keresztül, a kényelmi berendezésekig (például ablakemelő), és még sok más alkatrészben, illetve alrendszerben találunk Hall effektuson alapuló vezérlést [15].

4. fejezet

A rendszer specifikációi és architektúrája

4.1. Célkitűzés

A diplomamunkám tervezésénél az volt a cél, hogy egy olyan műszerfalat tervezzek, illetve kivitelezzenek, amely valós időben szimulálni tudja egy tényleges jármű műszerfalának működését. A műszerfal tervezésénél a következőket tartottam szem előtt:

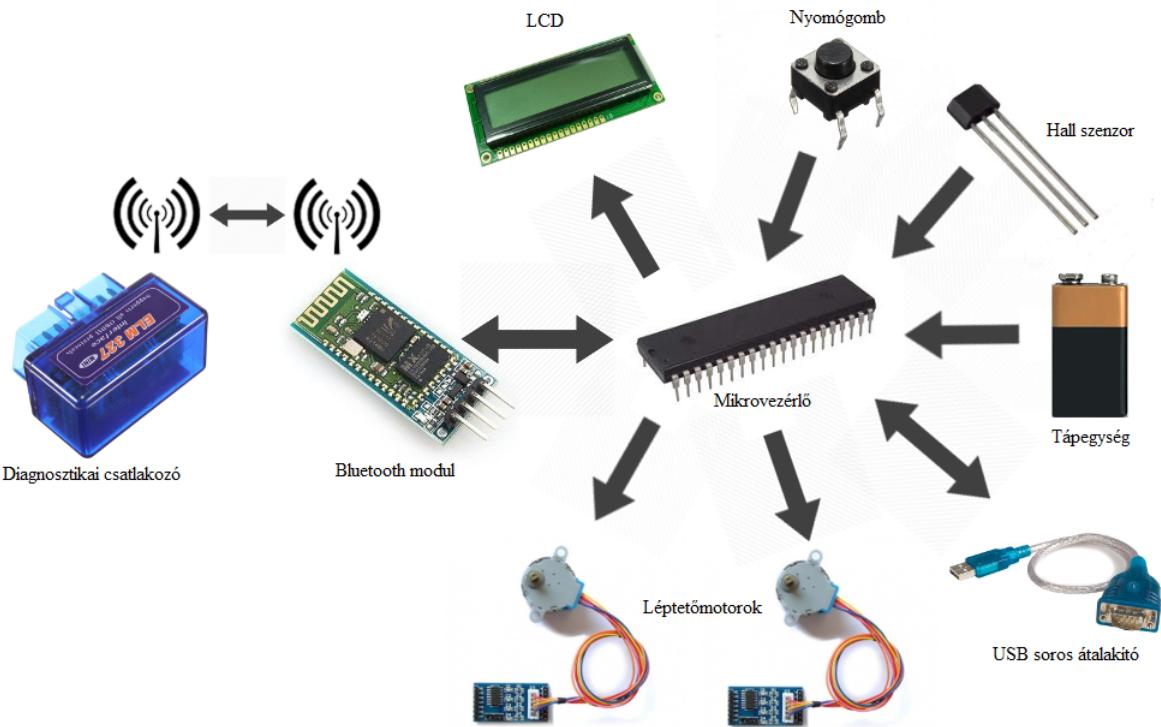
- USB-n keresztül, tárrezidens betöltő programmal való felprogramozhatóság
- a sebesség, és fordulatszám mutatók, egy valós jármű mutatóit követve, közel ugyanúgy viselkedjenek
- az elkészített műszerfal hordozható legyen, saját tápellátással
- a műszerfal tartalmazzon egy felhasználói felületnek megfelelő menürendszer
- a műszerfalat irányító menü, valamint néhány jármű paraméter egy LCD-re legyen kijelzve
- nyomógombokkal való navigálás a menüben
- a rendszer Bluetooth-on keresztül kommunikáljon a jármű diagnosztikai rendszerével

4.2. A rendszer architektúrája

Az általam készített rendszernek, lényegében két fontos része van. Az egyik a műszerfalat koordináló vezérlő egység, illetve a hozzá kötött perifériák, a másik pedig a gépjármű diagnosztikai rendszerével kommunikáló csatoló áramkör. A műszerfalat egy PIC18F4620-as mikróvezérlő irányítja, illetve hangolja össze a perifériák működését.

Az alfanumerikus kijelzőn (LCD) jelenik meg bekapsolás után a menü rendszer ahonnan navigálva tudjuk elindítani a járművel való kommunikációt, továbbá kiválasztva a műszerfalat indító opciót, a mutatók pozícióba állása mellett, még néhány információt is láthatunk az LCD-n a járműről (például akkumulátor feszültség).

A nyomógombok a menüben való navigáláshoz szükségesek. Ehhez három darab gombot használok, vagyis ennek megfelelően a menüben léphetünk fel, le, illetve kiválaszthatunk egy menüpontot.



4.1. ábra. A műszerfal tömbvázlata

A Hall szenzorok, a műszerórák mutatóinak, alaphelyzetbe, azaz a nullába való irányításához szükségesek. A műszerfalon a nullás vonalán lévő szenzor, megállítja a mutatót amikor érzékeli az arra felszerelt mágneset.

A rendszer működéséhez, egy 9V-os elem szolgáltatja az energiát. Az elem, kis méretéből adódóan hozzájárul a műszerfal hordozhatóságához.

A mikrovezérlő felprogramozását, egy tárrézidens betöltő program segítségével végzem (boot loader). Előzőleg egy erre a célra kifejlesztett programozó áramkör segítségével, betöljtük a mikrovezérlő programmemoriájába a boot loadert, innentől kezdve pedig ő végzi a feltöltést. A program beírása ezután egy soros adatátviteli csatornán történik (RS232), és ezért szükséges egy USB soros átalakító, hogy a PC-ről is programozni lehessen az eszközt.

A műszerórák mutatóinak mozgatását két léptetőmotor végzi, amelyekkel könnyen és pontosan egy pontba lehet vezérelni a mutatókat. A motorok meghajtásáról egy ULN2003a

meghajtó áramkör gondoskodik.

A műszerfal, a diagnosztikai csatoló áramkörrel, egy HC-05-ös Bluetooth modulon keresztül kommunikál. A modul, a mikrovezérlő UART perifériájához csatlakozva küldi, illetve fogadja az adatokat.

A jármű diagnosztikai rendszere, és egy soros kommunikációs protokoll között az ELM327 áramkör teremt kapcsolatot. Ezen az áramkör biztosítja a jármű adatainak kinyerését.

4.3. Hardver specifikáció

Az általam készített műszerfal, alapjába véve, úgy kell viselkedjen, mint egy valós járműben lévő műszerfal. Viszonylag nagy pontossággal, valós időben kell kijelezze a sebességet- és a fordulatszámot, továbbá pontosan kell a jármű néhány adatát kiírja a felhasználónak.

A legfontosabb bemeneti paraméter értelemszerűen a sebesség- és a fordulatszám, ezek után következik az akkumulátor feszültség, CO₂ kibocsátás, stb. Ezekkel a, hexadecimális bemeneti paraméterekkel dolgozik a mikrovezérlő. Továbbá megemlíteném bemeneti paraméterként a Hall szenzoroktól kapott analóg információt, ami a mutatók alaphelyzetbe állítását szolgálja, a nyomógombok bemeneti állapotait, illetve a tápot, amely a bemeneti feszültséget biztosítja.

A hardver (műszerfal) kimeneti paraméterei a léptető motorokat vezérlő digitális jelek, és az LCD meghajtáshoz szükséges paraméterek. Továbbá kimeneti paraméterek, a mikrovezérlő UART perifériájától a diagnosztikai csatoló áramkör fele utazó kérések is.

4.4. Szoftver specifikáció

A mikrovezérlőre írt program hangolja össze a műszerfal teljes működését. A program oly módon kell mindezt operálja, hogy a mikrovezérlő központi feldolgozó egysége, ne kerüljön soha tétlen állapotba. Minél gyorsabban dolgozza fel az adatot, és továbbítsa azt a perifériák felé. A program C programozási nyelv alatt íródott.

A szoftver bemeneti paramétere minden esetben az UART csatornán érkező hexadecimális adatok, amelyeket szoftver szinten, a felhasználó számára érthető alakba kell hozni. Miután beérkeztek az adatok, a szoftver ki kell szélektálja a hasznos adatot a pufferből, és ezután szoftveres átalakítások révén, a perifériák- és a felhasználó számára érthető formába kell alakítsa azt, úgy, hogy a vezérlő a program alapján a megfelelő digitális jeleket adja ki a perifériák fele. Továbbá a programnak, a megfelelő időzítésekről és megszakításokról is gondoskodnia kell, figyelembe véve a kommunikációs csatornán bejövő adatok- és az időzítők által generált megszakítások lekezelését.

A program kimeneti paramétere az adatok feldolgozása után, a mikrovezérlő fele adott utasítások sorozata, amely hatására az a megfelelő módon vezérli a perifériákat.

5. fejezet

Részletes tervezés, és gyakorlati megvalósítás

A diplomamunkám hardver része, egy mechanikai- és egy a szoftver irányította elektronikai rendszert foglal magába. A mechanikai tervezés esetében, egy fontos kritérium volt, hogy az általam készített műszerfal formailag hasonlítsa egy valós jármű műszerfalához, továbbá tartalmazza az megírt szoftvert vezérlő eszközt, és annak perifériáit. Mivel a rendszer mozgatható, figyelnem kellett a méretekre.

5.1. Hardvertervezés

5.1.1. Mechanikai tervezés

A műszerfalat alapját egy 24.5 x 21.5 cm-es műanyag lap alkotja, amire rákerült a műszerrákat, illetve az LCD-t tartalmazó váz, valamint a teljes elektronika (5.2 ábra).



(a)



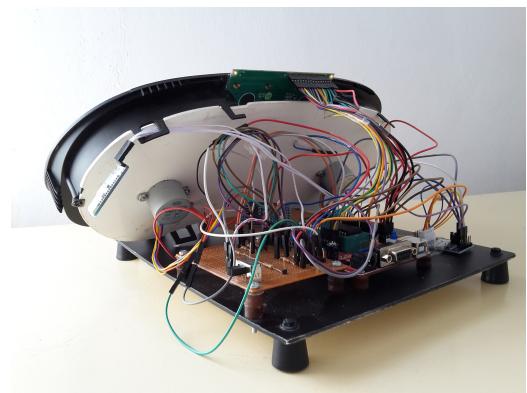
(b)

5.1. ábra. A mechanikai rendszer váza

A műszerórákat (sebesség és fordulatszám), léptetőmotorok valósítják meg amelyek egy, a vázhöz formált műanyag panelre vannak felerősítve. A panel egyik felén találjuk a motorokat, a másik felén pedig a sebességnak, illetve a fordulatszámnak megfelelő beosztásokat. A panel, a váz házához van illesztve, közéjük pedig az alfanumerikus kijelző került, lényegében ez a vázhöz van ragasztva.

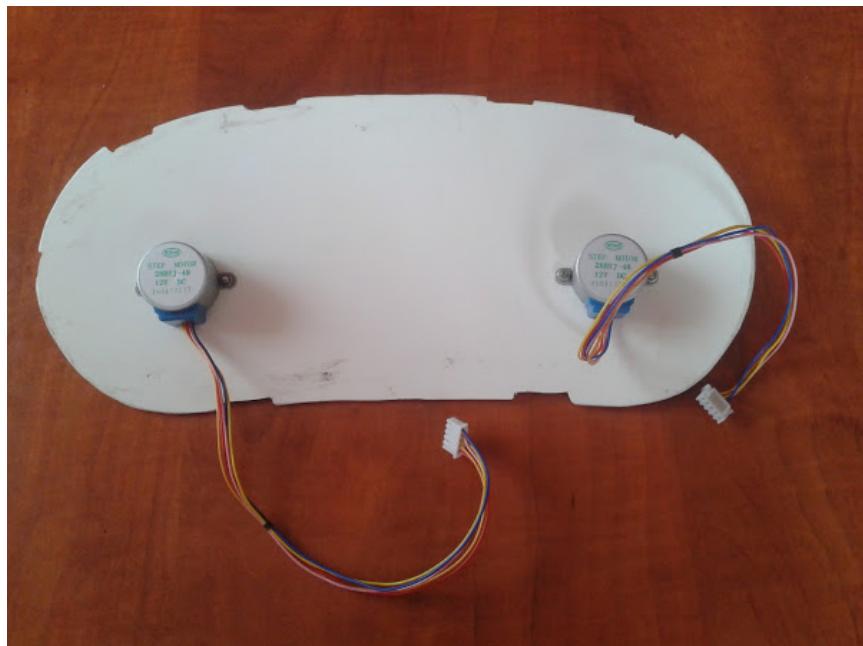


(a)



(b)

5.2. ábra. A teljes rendszer



(a)

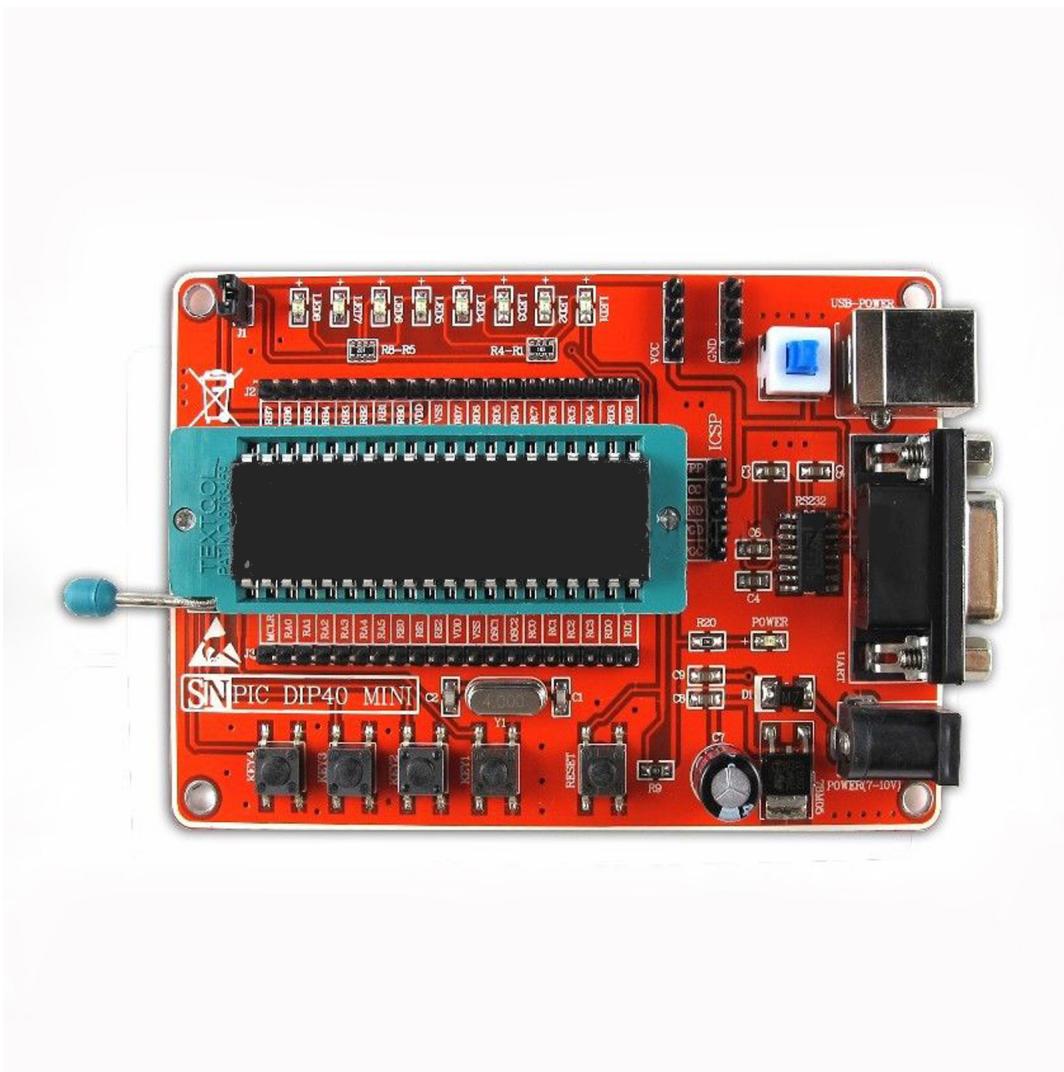


(b)

5.3. ábra. A motorokat, és a beosztásokat tartalmazó panel

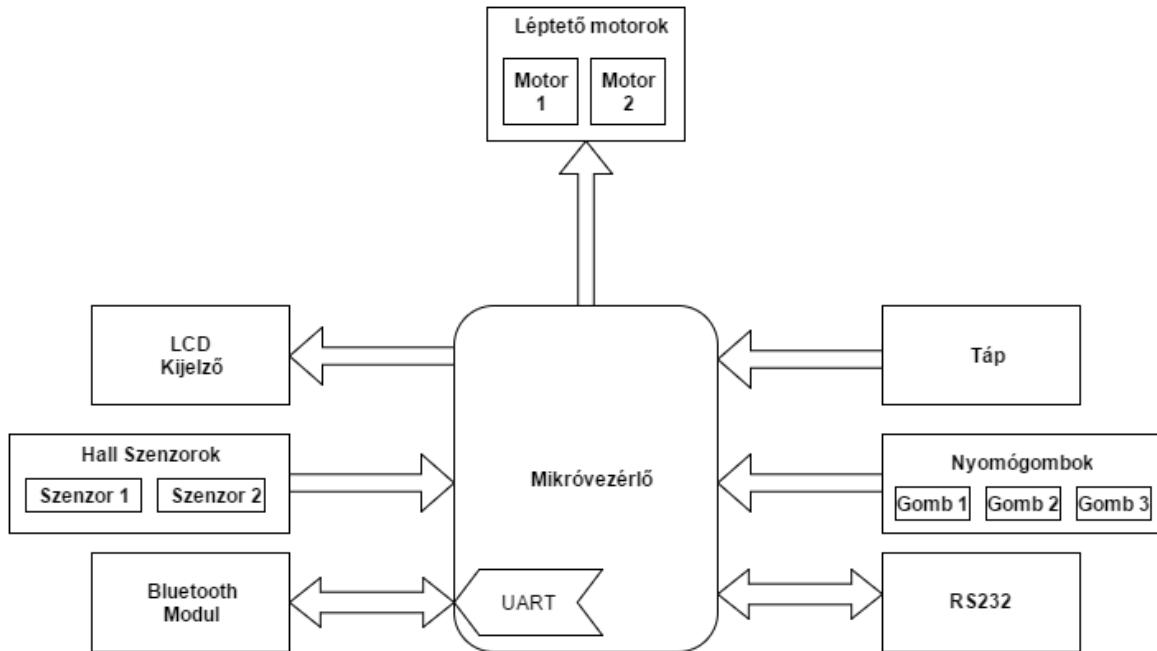
5.1.2. Elektronikai tervezés

A műszerfal elektronikai rendszere hangolja össze, illetve vezérli a teljes rendszert. Az elektronikát én terveztem, felhasználva a mikróvezérlős- és analóg elektronikai ismereteimet, a tervezésben a Proteus 8 Professional szoftver segített. Az elektronikai alrendszer tervezésének első lépése, egy mikróvezérlős fejlesztőlap beszerzése volt. Szempontok a megfelelő fejlesztőlap beszerzéséhez a következők voltak: legyenek gombok- és led-ek rajta, tudjam tápláni USB-ről, illetve egyéb külső tápról, és nem utolsó sorban, legyen lehetőségem USB-ről programozni a mikróvezérlőt. A könnyebb illesztés miatt, a mikróvezérlő lábai, legyenek tűsorokon keresztül ki vezetve, hogy így könnyen össze tudjam kötni azokat más áramkörökkel. Ezeket a szempontokat figyelembe véve a 5.4 ábrán látható PIC mikróvezérlőknek való fejlesztőlappal dolgozok.



5.4. ábra. Fejlesztőlap, PIC mikróvezérlők számára

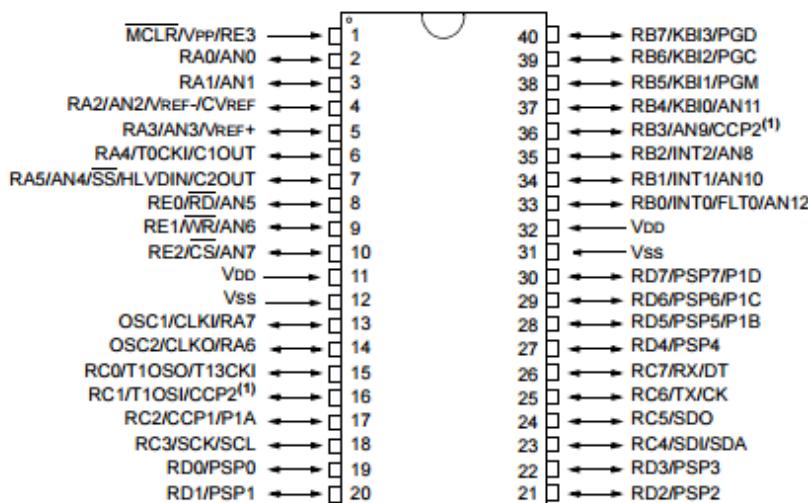
Nezzük meg egyenként az elektronikai rendszer komponenseit (5.5 ábra):



5.5. ábra. A műszerfal blokkdiagramja

A mikróvezérlő

A megvalósított műszerfálat irányító mikróvezérlő egy, a Microchip által gyártott, 18F családban lévő, 4620-as típusú PIC mikróvezérlő. A mikróvezérlő kiválasztásánál a fő szempontok azok voltak, hogy legyen benne UART sorok kommunikációs periféria, I/O portok, illetve legalább két időzítő áramkör. A PIC18F4620-as mikróvezérlő, 8 bites architektúrával rendelkezik, 8/16-bites időzítő áramkörökkel, továbbá 64Mb flash program memória van benne. A központi feldolgozó egység működéséhez szükséges órajelet, egy külső 4Mhz-es oszcillátor szolgáltatja, ami a tárrezidens betöltő program kapcsán, egy PLL áramkörrel fel van szorozva 4-el, vagyis a rendszer egy 16Mhz-es órajelen működik. Tudva azt, hogy egy utasítás/időzítő lépés 4 órajelet vesz igénybe, tudjuk azt hogy egy utasítás elvégzése 250nS (nanosecundum) [16].



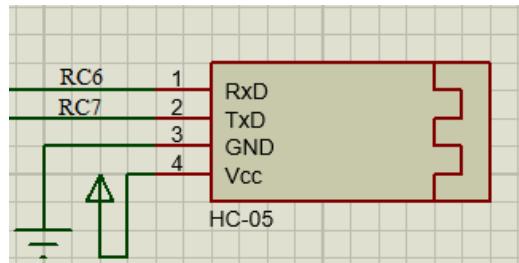
5.6. ábra. A PIC18F4620-as mikróvezérlő lábkiosztása

Elviekbén a mikróvezérlő, kéréseket küldd a diagnosztikai csatlakozó áramkörnek. majd a kapott válasz feldolgozása után vezérjeleket ad a léptető motorok, valamint az LCD fele.

A mikróvezérlő programozása egy tárrezidens betöltő program (bootloader) segítségével történik. A bootloader lényege, hogy előzetesen egy, erre a feladatra kifejlesztett, dedikált áramkörrel beírjuk ezt a betöltő programot a programmemoriába. Innentől kezdve, ő gondoskodik a programom fel- illetve betöltéséről, ami azt jelenti, hogy egy adatkapcsolati csatornára kapcsolódva (esetünkben RS232), kiolvassa a .hex fájlt, a sikeres olvasás után lekapcsolódik a csatornáról, az utasítás számlálót a beírt program kezdőcímére ugrasztja, és háttérbe vonul. A PC oldalon, egy tinyblWin nevű programmal olvasom be a kigenerált hex fájlt, majd kiválasztva a megfelelő USB portot, egy USB soros átalakító kábel segítségével, csatlakozva a vezérlő RS232 portjára, felprogramozom azt [11].

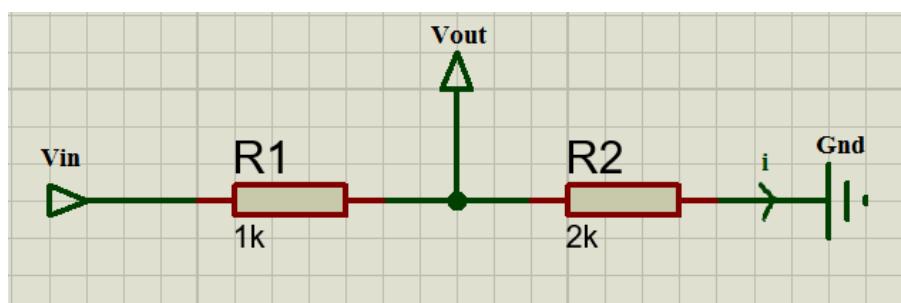
A Bluetooth modul

A mikróvezérlő, egy HC-05-ös Bluetooth modulon (3.10 ábra) keresztül kommunikál a diagnosztikai csatlakozóval (3.1 ábra). Azért választottam a HC-05 modult mert, első sorban fontos volt, hogy mester módban is tudjon működni, illetve kapcsolódni egy szolgá eszközre, továbbá egy kis, kompakt, könnyen kezelhető rendszer. Rengeteg AT parancs van hozzá, amivel konfigurálni lehet, de ezekről a szoftvertervezés alfejezetben fogok beszélni bővebben.



5.7. ábra. A HC-05 modul bekötése

A modulnak 4 lábat használom. Van egy GND, és egy Vcc láb, ezeket értelemszerűen a föld- illetve a tápfeszültségre kell kötni; továbbá van egy TX lába, amit a mikróvezérlő RX lábára kell kötni, és van egy RX lába amit a mikrovezérlő TX lábára kell kötni (5.7 ábra). A modul kommunikációs feszültségszintje 3.3 V (Volt), és mivel a mikróvezérlő 5 V-os feszültségszintet használ, ezért szükség volt egy feszültségesztő ellenállás párral beiktatására (5.8 ábra).



5.8. ábra. A feszültségesztő ellenállások

A feszültségesztő képletét az Ohm törvényből vezethetjük le:

$$I = \frac{U}{R} \rightarrow \begin{cases} i = \frac{V_{in}}{R1+R2} \\ V_{out} = i * R2 \end{cases} \quad (5.1)$$

Az 5.1 képletből következik tehát a feszültségesztő képlete:

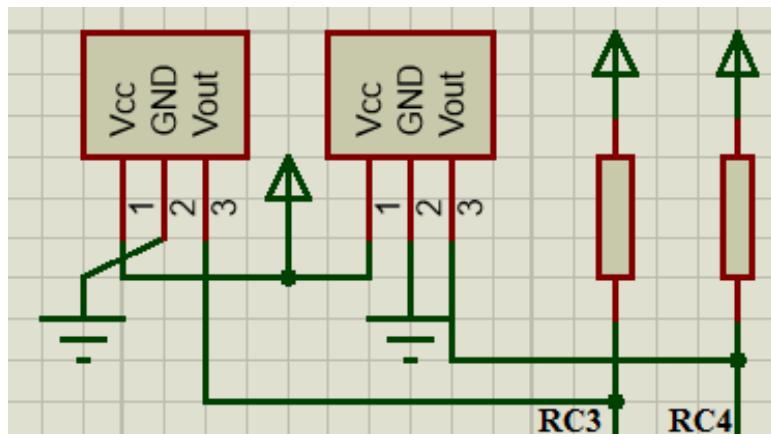
$$V_{out} = V_{in} * \frac{R2}{R1 + R2}$$

5.9. ábra. A feszültségesztő képlete

Az arányt kielégítő ellenállások bármelyike jó, én 1- illetve 2k-s ellenállásokat használtam (5.8 ábra).

Hall szenzorok

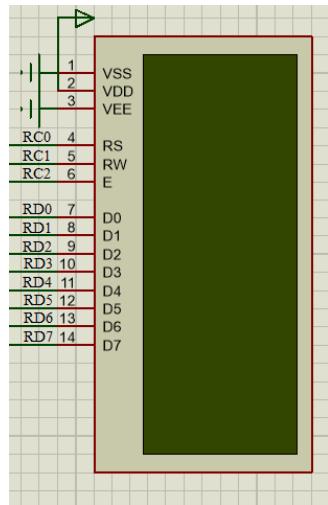
A Hall szenzorok kiválasztásának egyetlen szempontja az volt, hogy azok multipolárisak legyenek. A multipoláris Hall szenzorok lényege, hogy minden pólusra (észak és dél) reagálnak. A Honeywell által gyártott SS451A típusú Hall szenzort választottam. A szenzorok a léptető motorok mellett, a műanyag panelen helyezkednek el, pont a másik oldalon lévő beosztások, nullás számjegyei házához vannak beragasztva. A léptető motorok által vezérelt mutatók elejére, fel van ragasztva egy kicsi, viszont nagyon erős mágnes. A mutató amikor eléri a nulás számjegyet, a Hall szenzor érzékeli a mágneset, és leállítja a motorokat, vagyis a mutatók alaphelyzetbe kerültek.



5.10. ábra. A Hall szenzorok bekötése

LCD kijelző

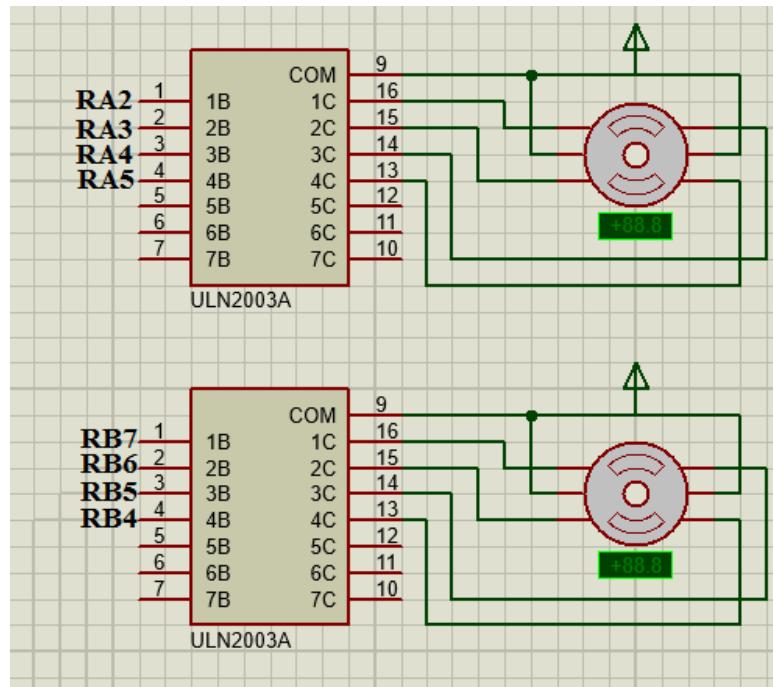
Az általam használt kijelző, egy alfanumerikus, folyadékkristályos kijelző aminek 4 sora van, soronként 20 karaktert képes megjeleníteni. Azért választottam karakterek kijelzésére fejlesztett képernyőt, mert csak számok, illetve betűk íródnak ki a felhasználók számára, és az alfanumerikus kijelzők a karakterek kiírását igényesen végeznek. Az LCD-t egy HD44780 vezérlő hajtja meg. A modul bekötése a 5.11 ábrán látható.



5.11. ábra. Az LCD modul bekötése

Léptető motorok

A műszerfal sebesség- illetve fordulatszám jelzőit, léptetőmotorokkal hajtom meg, mivel fontos, hogy viszonylag precízen a sebességnak, és a fordulatszámnak megfelelő pozícióba léptessem a mutatókat, továbbá fontos, hogy egy előírt pozícióba való lépés után, ne mozduljon ki onnan. Mivel a motorok nem kell nagy teljesítményt kifejtsenek, nem volt különösebb szempont a kiválasztásnál. Mivel a célnak bőven megfelel, én a 28BYJ-48 típusú, 4 fázisú, unipoláris léptető motort használom amelyek egy ULN2003A vezérlő hajt meg (3.3.1 alfejezet).



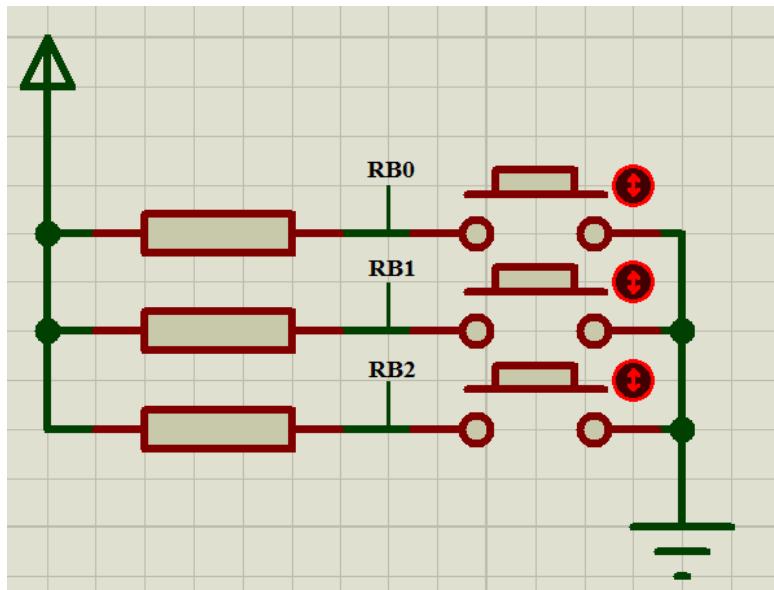
5.12. ábra. A léptető motorok bekötése

A táp

Mivel az általam tervezett rendszer esetében, szükséges hogy hordozható legyen, ezért fontos volt hogy egy kis tápellátás működtesse azt. A fejlesztés, illetve a tesztelés alatt, PC-ről, USB-n keresztül tápláltam a rendszert. Máskor egy 9V-os elem táplálja, aminek a kimeneti feszültségét egy regulátor lehozza a mikróvezérlő működési feszültségszintjére, azaz 5V-ra.

A nyomógombok

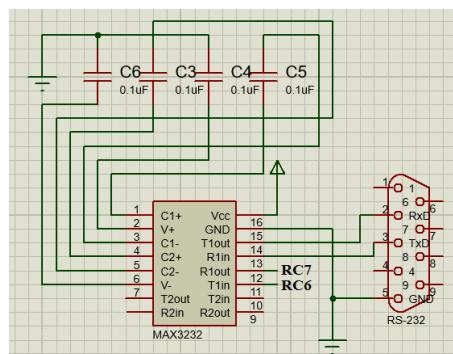
A nyomógombok, a főmenüben való navigáláshoz szükségesek. Három gombot használunk, a fel és a le navigáláshoz, valamint egyet a kiválasztott menübe való belépéshoz. A gombok lenyomás előtt pozitív tápfeszültségen tartják a mikróvezérlő lábait, majd lenyomás után földpotenciálra húzzák azokat. A gombok bekötése a 5.13 ábrán látható:



5.13. ábra. A nyomógombok bekötése

RS232 soros kommunikációs modul

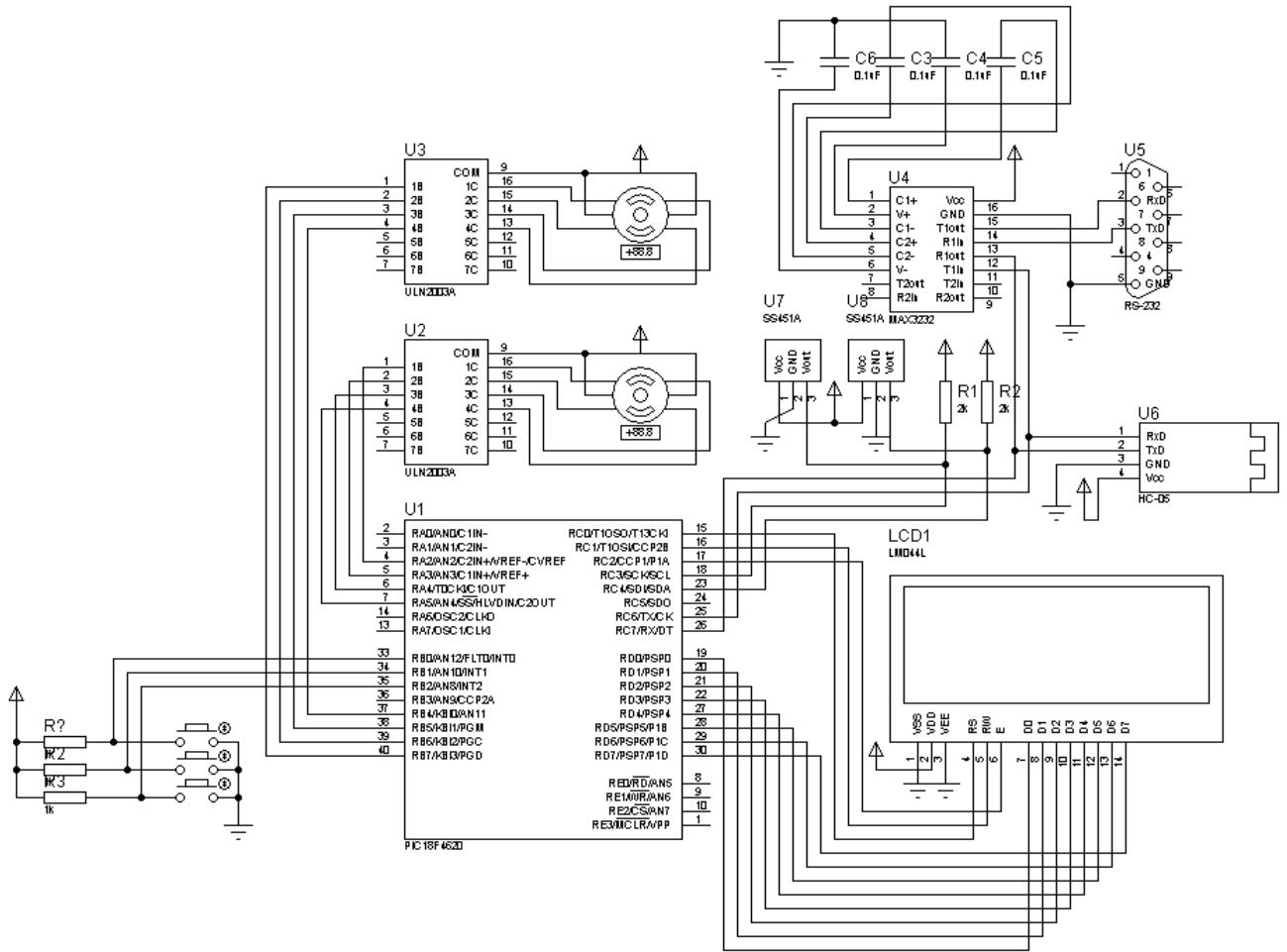
A mikróvezérlő felprogramozása, egy RS232-es modulon keresztül történik PC-ről. A PC oldali USB és a mikróvezérlőhöz kötött RS232-es periféria közti kapcsolatot egy USB soros átalakító kábel valósítja meg.



5.14. ábra. Az RS232-es modul

A programozás kezdetekor, a mikróvezérlőnek, szükséges egy RESET, hogy a regiszterek, illetve a perifériák egy jól meghatározott, kezdő állapotba kerüljenek.

Eddig megvizsgáltuk a rendszer komponenseit külön-külön, most lássuk a rendszer kapcsolási rajzát:



5.15. ábra. A rendszer kapcsolási rajza

5.2. Szoftvertervezés

A teljes rendszert irányító mikróvezérlőre írt programot, a Microchip által forgalmazott MPLAB IDE v8.92-es fejlesztő környezet alatt írtam C nyelven. A C nyelven megírt forráskódot, egy C18-as fordító program alakítja gépi kódú állománnyá (*.hex), amelyet a már említett tárrezidens betöltő program (bootloader) segítségével írok be a vezérlő programmemóriájába. A betöltő program egy soros kommunikációs vonalon kommunikál a számítógéppel, ahol egy Tiny Bootloader elnevezésű program segíti a betöltést. A szoftver tervezésénél arra törekedtem, hogy egységes időzítésű kódot írjak, kevés olyan függvényhívással amelyek valamilyen paramétert várnak. Minden változót statikusan deklaráltam, illetve inicializáltam, használva a C18-as fordító udata, idata és near pragmáit.

5.2.1. A megvalósított kommunikációs protokoll

A szoftver részletes ismertetése előtt, bemutatom a mikróvezérlő, és a diagnosztikai csatoló áramkör közti kommunikációt leíró protokolلت.

A diagnosztikai csatoló áramkör kapcsolatot teremt a jármű diagnosztikai rendszere, és egy kommunikációs csatorna között. A csatoló áramkör felhelyezését követően, az felismeri a jármű által használt kommunikációs protokolلت, és annak megfelelő alakba formálja a felé küldött kéréseket biztosítva ezzel egy könnyen használható interfész. Mivel én lényegében csak közvetetten férek hozzá a jármű diagnosztikai rendszeréhez, és adott egy jól meghatározott interfész, ezért csak az UART soros (Bluetooth) protokoll megtervezését kellett elvégeznem. A diagnosztikai csatoló áramkör értelmezi a soros porton kapott üzeneteket, és hibás kérés esetén nem juttatja el azt a jármű diagnosztikai rendszeréhez, hanem egy hibaüzenetet küld vissza.

A teljes kommunikáció karakter alapú, az adatok hexadecimális alakban kerülnek kiküldésre, illetve szintén ilyen alakban fogadja őket a mikróvezérlő. Fontos, hogy a jármű fele küldött üzenetek végén legyen egy CR (Carriage Return) karakter, mert azok csak így kerülnek értelmezésre. Az üzenet minden bájt-ja, bitenként került kiküldésre, illetve bitenként fogadja őket a mikróvezérlő.

A jármű felől érkező csomag végét egy ">" karakter jelzi. A megszakítást kezelő függvény ennek a karakternek a beérkezését figyeli.

A kiküldött csomag tartalmazza a kérés típusát (2.2.1 alfejezet), illetve a kívánt paraméter PID kódját. Az összes ilyen kód megtalálható a [17] weboldalon.

A fogadott csomag tartalmaz egy, a kérés sikeres kiszolgálását igazoló bájt-ot (0x41), utána a kérésben szereplő PID kódot ami szintén egy bájt, majd utána a magát a paraméter értékét, ami lesz számunkra a hasznos adat. A hasznos adat egy- vagy két bájt lehet az adat decimális értékétől függően.

5.2.2. A Bluetooth modul konfigurációja

A Bluetooth kommunikációhoz használt perifériaként a már említett HC-05 ös modult használom (3.4 fejezet). A modulhoz nagy konfigurációs parancslista áll rendelkezésre. Ezekkel az "AT" parancsokkal, úgy konfiguráltam be a modult, hogy az először is mester funkcióba működjön, illetve betáplálás után automatikusan felismeri az ELM327-es csatoló áramkört, és kapcsolódik hozzá.

A konfigurálás során, a modult "AT" parancs módba kell kapcsolni, ezután értelmezi a küldött konfigurációs parancsokat. A konfigurálást számítógépes terminál programból csináltam, illetve egy USB/TTL áramkört használtam ehhez. A konfiguráláshoz a következő parancsszektenciát futtattam le sorban:

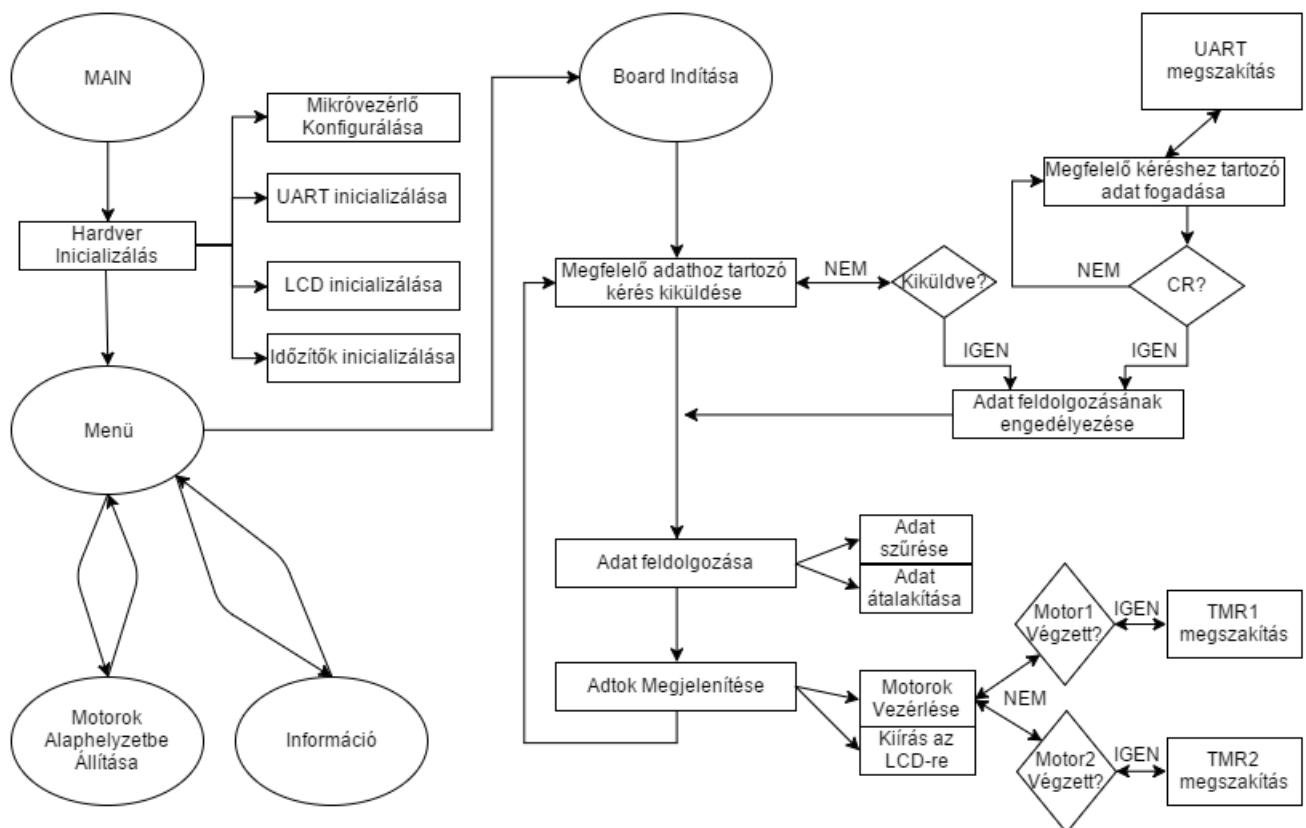
1. AT - helyes kommunikációt igazoló kérés
2. AT+ORGL - alapértelmezett állapotba rakja a modult

3. AT+RMAAD - törli a párosított eszközöket
4. AT+ROLE=1 - mester módba állítja a modult
5. AT+RESET - újraindítja a modul, ami mostmár mesterként funkcionál
6. AT+CMODE=1 - csatlakozás bármilyen címre
7. AT+INIT - modul inicializálása
8. AT+INQ - környező bluetooth eszközöket keres, és kilistázza a címüket
9. AT+PAIR=<ELM327 cím>,20 - párosítja a mester modult a megadott című szolga eszközökkel
10. AT+BIND=<ELM327 cím> - összeköti a címeket
11. AT+CMODE=0 - csak az összekötött címhez csatlakozzon
12. AT+LINK=<ELM327 cím> - csatlakozik a modul a szolga eszközre

Lefuttatva a fenti parancsokat, a modul ezután automatikusan az ELM327 csatolóáramkörről fog csatlakozni.

5.2.3. A mikróvezérlőre írt szoftver komponenseinek magyarázata

Lássuk előbb a mikróvezérlőre írt szoftver folyamatábráját (5.16 ábra):



5.16. ábra. A szoftver egyszrűsített folyamatábrája

Hardver Inicializálás

A program indulásakor, először a rendszer inicializálása, illetve konfigurálása hajtódik végre. Ez a program elején egyszer hajtódik végre, mielőtt az továbbhaladna a főmenübe.

A mikróvezérlő konfigurálása: itt állítódnak be a ki/bemeneti portok (irányai, és kezdeti értékei), illetve itt engedélyezem a magas, valamint az alacsony prioritású megszakításokat.

```

InitLCDPins();           //LCD-hez kapcsolódó pinék irányainak beállítása
InitSWPins();            //Nyomógombokhoz kapcsolódó pinék irányainak beállítása
InitHALLPins();          //Hall szenzorokhoz tartozó pinék irányainak beállítása
InitMOTORPins();         //Motorokhoz tartozó pinék irányainak beállítása
  
```

```

RCONbits.IPEN = 1;        //a megszakítás szintek engedélyezése
INTCONbits.GIEH = 1;       //magas prioritású megszakítás engedélyezése
INTCONbits.GIEL = 1;       //alacsony prioritású megszakítás engedélyezése
  
```

UART inicializálás: az UART inicializálás során állítódnak be a 38400 bps átviteli sebeségű, soros aszinkron kommunikációhoz szükséges feltételek.

```
TRISCbits.TRISC6 = 0;      //TX láb kimenetként való beállítása
TRISCbits.TRISC7 = 1;      //RX láb bemenetként való beállítása

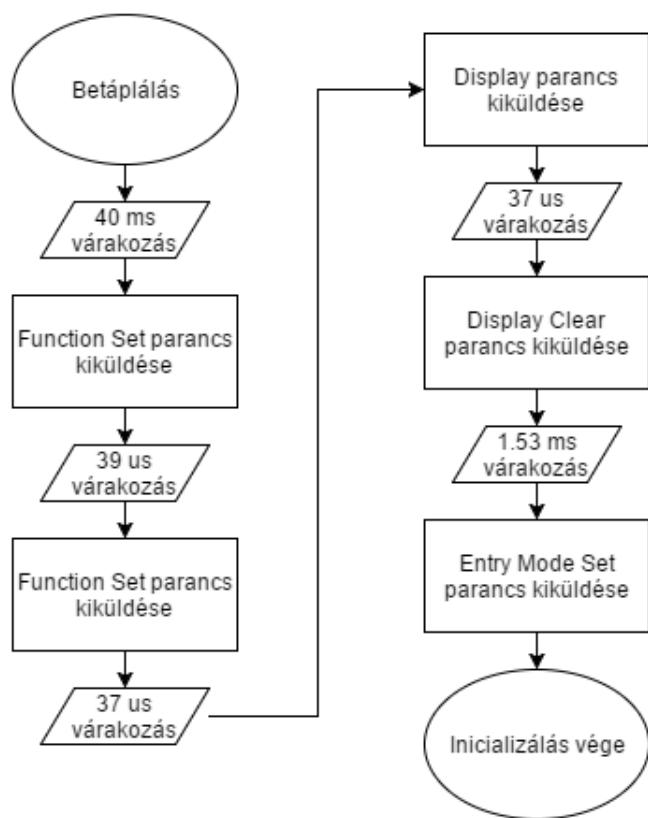
TXSTAbits.TX9 = 0;        //8-bites átvitel beállítása
TXSTAbits.TKEN = 1;        //küldés engedélyezése
TXSTAbits.SYNC = 0;        //aszinkron kommunikáció beállítása
TXSTAbits.BRGH = 1;        //magas sebességű kommunikáció beállítása
RCSTAbits.SPEN = 1;        //TX és RX lábak soros portként való beállítása
RCSTAbits.RX9 = 0;        //8-bites vétel beállítása
RCSTAbits.CREN = 1;        //folyamatos aszinkron fogadás engedélyezése
BAUDCONbits.BRG16 = 0;    //8-bites átviteli sebesség beállítása (csak SPBRG regiszter)
SPBRG = 25;                //38400 bps sebességű átvitel beállítása

PIE1bits.TXIE = 0;         //UART küldés megszakításának tiltása
PIE1bits.RCIE = 1;         //UART fogadás megszakításának engedélyezése
IPR1bits.RCIP = 1;         //UART fogadás, mint magas prioritású megszakítás beállítása
```

Időzítő inicializálás: A léptetőmotorok meghajtását időzítő áramkörökkel végzem. Az adat beérkezését, és feldolgozását követően egy jelzést küldve az időzítőknek, azok a megfelelő pozícióba vezérlik a mutatókat. Az alábbi ábrán látható a timer0 inicializálása. A timer1-et is hasonló módon inicializálom.

```
void TMRO_init(void)
{
    T0CON = 0x88;           //timer0-t irányító regiszter beállítása
    TMROH = HIGHTMRO;       //magas helyiértékű bájt - az időzítéshez
    TMROL = LOWTMRO;        //alacsony helyiértékű bájt - az időzítéshez
    INTCON2bits.TMROIP = 0; //alacsony prioritású megszakítás a túlcordulásnál
    INTCONbits.TMROIE = 0;  //túlcordulás megszakításának tiltása
    INTCONbits.TMROIE = 1;  //túlcordulás megszakításának engedélyezése
}
```

LCD inicializálása: ahhoz, hogy az LCD helyesen működjön, illetve alaphelyzetbe kerüljön, a betáplálás után szükséges mindenkorán a 5.17 ábrán látható rutint lefuttatni.



5.17. ábra. Az LCD inicializálásának lépései

Ezeket a lépeket a következő függvény hajtja végre:

```

void LCD_init(void)
{
    Delay10KTCYx(20);
    LCD_cmd(MODE_8BIT);
    Delay1KTCYx(1);
    LCD_cmd(MODE_8BIT);
    Delay1KTCYx(1);
    LCD_cmd(DISPLAY_ON);
    Delay1KTCYx(1);
    LCD_cmd(CLRSCR);
    Delay10KTCYx(1);
    LCD_cmd(ENTRY_MODE);
}

```

Az inicializálások, illetve a portok konfigurálását követően egy kis késleltetés következik, majd meghívódik a főmenüt megjelenítő függvény.

Felhasználói felület megjelenítése

A felhasználói felület, egy kis menüből áll, amelyben fel, és le tudunk navigálni. A ki-választott menüpontnál az "ENTER" gomb lenyomásával, a menüponthoz tartozó függvény meghívódik. A menürendszer a következő struktúra valósítja meg:

```

typedef const struct MenuStructure
{
    const char *text;           //menüpont neve
    unsigned char numberOfMenupoints; //hány menüpont van az aktuális menüben
    unsigned char up;           //felfele gomb lenyomásával melyik menüpontba kerülünk
    unsigned char down;          //lefelé gomb lenyomásával melyik menüpontba kerülünk
    unsigned char enter;         //enter gomb lenyomásával melyik menüponton maradunk
    void (*functionPointer) (void); //függvénymutató deklarálása
} MenuBuild;

```

A menüt egy *MenuBuild* típusú tömb tartalmazza, amely alapértelmezetten fel van töltve a megfelelő értékekkel. A tömb végigjárásával, illetve megfelelő struktúra elemekre való hivatkozással tudjuk manipulálni a látható menüt. A tényleges menü megjelenítését az alábbi függvény végzi:

```

void BrowseMenu(void)
{
    while(1)
    {
        ShowMenu();

        if (Debounce_SW1())
        {
            selected = menu[selected].up;
        }
        if (Debounce_SW2())
        {
            selected = menu[selected].down;
        }
        if (Debounce_SW3())
        {
            break;
        }

        Delay1KTCYx(10);
    }
    if(menu[selected].functionPointer != 0)
    {
        selected = menu[selected].enter;
        menu[selected].functionPointer();
    }
}

```

A menüpont kiválasztásáig egy végtelen ciklusba, a *ShowMenu()* függvénytel megjelenítem a képernyőn látható aktuális szöveget, illetve a ciklusban tovább haladva, figyelem a nyomógombokat. A *Debounce_**() függvények elvégzik a gomb lenyomása utáni pergésmentesítést, illetve jelzést adnak annak lenyomásáról. A *selected* változó, minden az aktuálisan kiválasztott menüpont indexét fogja tartalmazni. Az "ENTER" gomb lenyomása után, a program kilépik a végtelen ciklusból, és a függvénymutatón keresztül meghívódik a menüponthoz tartozó függvény. A végtelen ciklusból azért lépek ki egy menüpont kiválasztása után, hogy a soron következő műveletek alatt a központi feldolgozó egység ne maradjon benne ebben a végtelen ciklusban, illetve ne tesztelje továbbra is a nyomógombok állapotát, csak amikor a főmenüben vagyunk. Ezzel is gyorsítva a további feladatok elvégzését.

Az LCD-re való kiírást az alábbi függvény végzi:

```

void LCDWriteMenu(int lineCount, unsigned char *text)
{
    unsigned char i;
    unsigned char lcdOffset = 0;
    unsigned char x = 0;

    if (lineCount > (LCDMAXY-1)) lineCount = 0;

    switch(lineCount)
    {
        case 0: BF=0;LCD_cmd(LCD_LINE1);break;
        case 1: BF=0;LCD_cmd(LCD_LINE2);break;
        case 2: BF=0;LCD_cmd(LCD_LINE3);break;
        case 3: BF=0;LCD_cmd(LCD_LINE4);break;
        default:BF=0;LCD_cmd(LCD_LINE1);
    }

    lcdOffset = strlen(text);

    if (lcdOffset > LCDMAXX) lcdOffset = LCDMAXX;
    for(i=lcdOffset; i ; i--)
    {
        LCD_data(*text);
        text++;
    }
}

```

Lényegében az *LCDWriteMenu(int, unsigned char*)* függvény használom a menüpontok, és egyéb adatok kiírására is. A függvénynek, első paraméterként meg kell adjuk, hogy hányadik sorba szeretnénk kiírni a szöveget, második paraméterként pedig magát a kiírandó szöveg. Mivel az LCD 4x20-as ezért nem lehet csak 4 (0-3) sorba, illetve csak 20 darab karakterből álló szöveget kiíratni. Miután megadtuk a paramétereket, a függvény az LCD-nek kiküldi a megfelelő sor kezdő címére való ugrás parancsot, és karakterenként, az LCD adat portjára, kiküldi a 8 bites adatcsomagot.

A műszerfal elindítása

A menü fő menüpontja a *Start Board* ami lényegében elindítja a kommunikációt a jármű és a mikróvezérlő között. A *StartBoard()* függvénybe lépve egyszer kiíratom a műszerfal működése közbeni, kijelzett adatok nevét, illetve azok mértékegységék a kijelzőre, ezután egy végtelen ciklus következik amelyben történik a feladatok összehangolása.

```

void StartBoard()
{
    LCD_cmd(CLRSCR);
    LCDWriteMenu(0, obd1);
    LCDWriteMenu(1, obd2);
    LCDWriteMenu(2, obd3);
    LCDWriteMenu(3, obd4);
    while(1)
    {
        SendRequests();
        HandleReceivedData();
        ParseData();
    }
}

```

A jármű diagnosztikai rendszerétől, négy adatot kérek le a kommunikáció során: a sebességet, fordulatszámot, a hűtőfolyadék hőmérsékletét, illetve a levegőszűrőn beszívott levegő hőmérsékletét. Az utóbbi két paramétert celsius fokban íratom ki. Mivel az ELM, hexadecimális karaktereket vár, ezért a kéréseket egy *unsigned char* típusú tömbben tárolom, amelyből az adatok karakterenként lesznek kiküldve. Mint említettem fontos, hogy a kérés végén ott legyen az CR karakter. így néznek ki a kérések:

```

unsigned char COT[] = "0105\r"; //hűtőfolyadék hőmérséklete
unsigned char RPM[] = "010C\r"; //fordulatszám
unsigned char KPH[] = "010D\r"; //sebesség
unsigned char IAT[] = "010F\r"; //beáramló levegő hőmérséklete

```

A végtelen ciklus első függvénye az *SendRequests()* gondoskodik a megfelelő paraméterekhez tartozó kérések kiküldéséről. A *temp_switch* ideiglenes változó gondoskodik az adat kérések, - illetve az adat fogadások szinkronizálásáról.

```

void SendRequests( void )
{
    if(!REQUESTSENT)
    {
        if (temp_switch == 0)
        {
            SendRequest(RPM);
            REQUESTSENT = TRUE;
        }
        else if(temp_switch == 1)
        {
            SendRequest(KPH);
            REQUESTSENT = TRUE;
        }
        else if(temp_switch == 2)
        {
            SendRequest(COT);
            REQUESTSENT = TRUE;
        }
        else if(temp_switch == 3)
        {
            SendRequest(IAT);
            REQUESTSENT = TRUE;
        }
    }
}

```

A második függvény a *HandleReceivedData()* gondoskodik a lekért adat feldolgozásáról, vagyis a *temp_switch*-el jelzem hogy melyik adat volt lekérve, és ha volt kérés kiküldve, valamint érkezett be csomag, akkor a *temp_switch* szerint, a megfelelő változóba beleteszem azt. Mint említettem, a beérkezett csomag két bájtja csak a számomra hasznos adat, ezért egy sima index címzéssel kiszedem azt a két bájtot a pufferból, ezután átalakítva *int* típussá, tovább küldöm a kimutatást végző függvénynek, illetve az adatok egy részét helyben kiíratom az LCD-re.

```
void HandleReceivedData(void)
{
    if(REQUESTSENT && SERIALRCRECEIVED)
    {
        if(temp_switch == 0)
        {

            tempBuff[0] = rcBuffer[11];
            tempBuff[1] = rcBuffer[12];
            tempBuff[2] = rcBuffer[14];
            tempBuff[3] = rcBuffer[15];

            rpm = ((xtoi(tempBuff))/4);
            if(rpm < 1000)
            {
                LCDWriteMenu(0, obd1);
            }

            sprintf(lcd_buffer, "%d", rpm);
            LCDWriteMenu(0, lcd_buffer);
            memset(tempBuff, 0, sizeof tempBuff);
            temp_switch = 1;
            RPMREADY = TRUE;

        }
        else if(temp_switch == 1)
        {
            -
            -
            -
        }
    }
}
```

A ciklus soron következő függvénye a *ParseData()*, amely a léptetőmotorok vezérléséhez szükséges számításokat végző függvényeket hívja meg. Vizsgálja, hogy jött-e be fordulatszámot, vagy sebességes leíró adat, és ha igen, meghívja a bejött adatnak megfelelő, pontos lépésszámot kiszámoló függvényt.

```
void ParseData(void)
{
    if(RPMREADY)
    {
        ControlMotorRPM();
        RPMREADY = FALSE;
    }
    if(KPHREADY)
    {

        ControlMotorKPH();
        KPHREADY = FALSE;
    }
}
```

Megszakítás kezelés

A szoftverben egy magas, illetve egy alacsony prioritású megszakítást is beimplementáltam. A magas prioritású megszakítás az UART megszakítás kéréseit kezeli, míg az alacsony prioritású az időzítők túlcordulásait.

Magas prioritású megszakítást lekezelő függvény, amely egy pufferbe rakja be, 8-bitenként a soros aszinkron csatornán érkező adatot, figyelve közbe, hogy nem-e jött be a csomag végét jelző (0x3E hexadecimális) karakter. Ha ez bejött, jelzi a *HandleReceivedData()* függvénynek, hogy feldolgozhatja az adatot.

```
void ISRHighHandler( void )
{
    if ( (PIR1bits.RCIF) && (PIE1bits.RCIE) )
    {
        if(!SERIALRCRECEIVED)
        {
            serialRcByte = RCREG;
            if(serialRcByte == 0x3E)
            {
                temp_index = 0;
                SERIALRCRECEIVED = TRUE;
            }
            else
            {
                rcBuffer[temp_index] = serialRcByte;
                temp_index++;
            }
        }
    }
}
```

Az alacsony prioritású megszakításokat lekezelő függvény, az időzítők kiszolgálásáért felelős, amelyek a léptetőmotorokat vezérlik. Ha egy beérkezett adat, elérkezett a kijelzés fázishoz, ezt jelez a megszakítást lekezelő függvénynek, amely a megfelelő léptetőmotort az előírt pozícióba vezérli, majd letiltja az adott motorhoz tartozó, időzítő megszakítást, amelyet majd egy újabb beérkezett adat fog engedélyezni. A vezérlés kulcsfontosságú mozzanata a pozíció helyes felismerése, amelyet jelzőbitekkel oldottam meg. Az alábbi kód részletben a fordulatszámot vezérlő motor megszakítás lekezelése látható. A (a) fölötti kód részlet, a motor pozitív, azaz az óra mutató járásával megegyező irányba való vezérlést végzi, míg a (b) fölötti, a motor negatív irányba való hajtását.

A sebesség vezérlése is, hasonló módon történik.

```

if(RPMMOTORPOS[0])
{
    if (motordata_index == 8)
    {
        motordata_index = 0;
        counterTMRO++;

        if(counterTMRO >= numstep_rpm)
        {
            INTCONbits.TMROIE = FALSE;
            counterTMRO = 0;
            RPMMOTORPOS[2] = FALSE;
            FLAGS[0] = TRUE;
        }
        else
        {
            INTCONbits.TMROIE = TRUE;
        }
    }
    else
    {
        MOTORPORTRPM = RPMMOTORDATA_N[motordata_index];
        motordata_index++;
        INTCONbits.TMROIE = TRUE;
    }
}
else
{
    if(RPMMOTORPOS[1])
    {
        if (motordata_index == 8)
        {
            motordata_index = 0;
            counterTMRO++;

            if(counterTMRO >= numstep_rpm)
            {
                INTCONbits.TMROIE = FALSE;
                counterTMRO = 0;
                RPMMOTORPOS[2] = FALSE;
                FLAGS[0] = TRUE;
            }
            else
            {
                INTCONbits.TMROIE = TRUE;
            }
        }
        else
        {
            MOTORPORTRPM = RPMMOTORDATA_N[motordata_index];
            motordata_index++;
            INTCONbits.TMROIE = TRUE;
        }
    }
}
}

(a) (b)

```

Egyéb függvények

A kéréseket a *SendRequest(char * request)* függvényel küldöm ki. A függvény egy karakterláncot vár paraméterül és karakterenként küldi ki azokat.

```

void SendRequest (char * request)
{
    while (*request)
    {
        while (!PIR1bits.TKIF); //vár amíg a küldő regiszter üres lesz
        TXREG=*request++; //karakterenként kiküldi az adatot
    }
}

```

A következő szintén jelentős függvény a *int xtoi(char *hexstring)* amely egy hexadecimális karakterekből álló tömböt vár, amelyet átalakítva *int* típusként téríti vissza. Mivel minden adat hexadecimális alakban érkezik a járműtől, ezért fontos hogy azokat megjeleníthető formába tudjam alakítani.

6. fejezet

A rendszer felhasználása

A diplomamunkám felhasználását tekintve, egy jó alapot ad annak az olvasónak, aki az autóipari kommunikációt szeretné megismerni, illetve aki a műszaki diagnosztikai elvek iránt érdeklődik. A diplomamunkám készítése közben igyekeztem jól megismerni ezeket az elveket, protokollokat és összefoglalni azokat a gondolatokat, amelyek nagy-vonalakban meghatározzák ennek az iparának a szisztemámáját.

A megvalósított mikróvezérlős rendszerrel pedig jól lehet szimulálni egy valós műszerfal, üzem közbeni működését, illetve néhány olyan adatot kijelezni a kocsiról, amelyeket hétköznapi használat során nem lát a felhasználó. Továbbá a mikróvezérlőre írt szoftver segítséget nyújt, olyan projektekhöz, amelyekhez soros aszinkron kommunikáció szükséges, léptetőmotorok vezérlését kell megoldják, illetve szükségszerű alfanumerikus LCD-re való kiírás.

7. fejezet

Üzembe helyezés és kísérleti eredmények

Az elkészített műszerfalam tesztelés közben USB-ről volt táplálva, vagyis 5V-al, különben pedig 7-10V feszültségű táppal is be lehet táplálni. Mivel a rendszer hordozható, én egy 9V-os alkáli elemet használok a betápláláshoz. A betáplálás után, a mikróvezérlő inicializálja a perifériákat, majd bekapcsolva az LCD-t, a főmenübe lép. Ezután a három nyomógomb segítségével tudunk tovább navigálni a menüben. A legelső fontos dolog ahhoz, hogy a rendszer megfelelően működjön, alaphelyzetbe (nullába) kell hozzuk a műszerórák mutatóit. Ezt a legelső menüponttal tehetjük meg. Majd miután a mutatók alaphelyzetbe álltak, elindíthatjuk a műszerfalat. Ez előzetesen a csatoló áramkör felhelyezését, illetve a jármű begyújtását követeli.

7.1. A rendszer tesztelése

A rendszer tesztelése, két fázisban történt. Az első fázisban az UART soros kommunikációt teszteltem le, mivel egy fontos része a rendszeremnek. A teszteléshez különböző terminal programokat, illetve USB/TTL kiegészítő áramkört használtam, amelyekkel meg tudtam győződni, hogy az adatok helyesen küldődnek ki, illetve helyesen lesznek fogadva a vezérlő által.

A második fázisban pedig az összerakott rendszert teszteltem, valós jármű segítségével. Miután felhelyeztem a csatoló áramkört, a jármű diagnosztikai csatlakozójára, begyújtottam a járművet, illetve a műszerfalam bekapcsolása után, elindítottam a mikróvezérlőre írt programot. A következő képeken látható, hogy valós időben, a jármű műszerfalával párhuzamosan, az én műszerfalam is pontos értékeket mutat az LCD-n, illetve a fordulatszám (és sebesség) mutatóim is pontosan beállnak a megfelelő pozícióba, követve a jármű mutatóit.



(a)



(b)

7.1. ábra. A rendszer tesztelése**7.2. ábra.** A rendszer tesztelése

A rendszert négy különböző márkájú autón teszteltem, amelyekből 3 gázolajjal, 1 pedig benzinnel működött. Mind a négy jármű más-más kommunikációs protokollt használt, de az ELM327-et beállítva automata módra, felismerte a járművek által támogatott protokollet. A különbség, az adatok küldésének gyorsaságában volt: a CAN protokollt támogató gázolaj meghajtású jármű, gyorsabban küldte az adatokat, mint a KWP2000-et támogató, benzines jármű.

8. fejezet

Következtetések

A diplomamunkám során sikerült az autóipari kommunikációs protokollokat megismernem, illetve ezzel párhuzamosan sikerült a műszaki diagnosztika működési elveit tisztázzam, betekintést nyerve ezzel a napi szinten használt járművek szenzorainak- és jeladónak világába.

A feladat segített tapasztalatot szerezni, egy valós mikróvezérlős alkalmazás tervezésében, illetve a tervezési lépések követése általi kivitelezésben, jól imitálva ezzel egy valós alkalmazás esetében, azoknak a műszaki feladataknak az elvégzését amelyeket, egy számítástechnika- (és automatizálás) szakos hallgatóra bízhatnak.

A mikróvezérlő írt szoftvert két megközelítésből írtam meg. Az első megközelítésben implementált kódban sok olyan rés volt ahol a központi feldolgozó egység tétlen állapotban állt, vagy várt valamely adatra, illetve valamely függvény befejeződésére. Ez sok időt vett igénybe, késleltetve az adatcserét, illetve a kijelzést. Mivel így lassabban álltak be a műszerfalak mutatói a megfelelő pozícióba, ezért úgy csoportosítottam át a kódot, hogy ezeket a lehető legjobban kiküszöböljem, illetve a sebesség növeléséhez bekapcsoltam még néhány perifériát (időzítő), és megszakítást.

8.1. Megvalósítások

Sikerült egy olyan rendszert létrehoznom, amely segítségével a diagnosztika működési elveit bizonyítani tudtam, vagyis megvalósítottam egy mikróvezérlős rendszert, amely egy diagnosztikai áramkör segítségével, közvetetten kapcsolódni tud a jármű diagnosztikai rendszeréhez, és adatokat lekérni onnan, illetve a lekért adatok feldolgozását követően, ezeket kijelzi a felhasználók számára. Ez a kommunikáció egy stabil Bluetooth protokoll felépítését követelte, amelyen keresztül a mikróvezérlő nagy sebességgel képes adatot cserálni a diagnosztikai áramkörrel.

A mikróvezérlős rendszer egésze, a perifériákkal, és a kiegészítő elektronikával együtt, egy szintén általam tervezett, műszerfalat szimuláló, hordozható mechanikai rendszerbe vannak beépítve.

8.2. Hasonló rendszerekkel való összehasonlítás

Valós jármű műszerfalával összehasonlítva, elmondhatom, hogy az én műszerfalam is közel pontosan mutatja a sebességet, fordulatszámot, illetve a különböző kiegészítő adatot. A jármű műszerfalán látható adatok jelennek meg az én rendszeremen is.

8.3. További fejlesztési lehetőségek

A további fejlesztési lehetőségek esetében, elsősorban megemlíteném a kommunikációs hatótávolság növelését. Mivel most egy Bluetooth protokoll valósítja meg a kommunikációt, ez azt feltételezi, hogy az általam készített műszerfal, közel legyen a járműhöz, viszont egy nagyobb hatótávú kommunikációs protokoll mint például a Wifi, sokat segítene a rendszer tesztelésében, főként amikor a jármű sebességére vonatkozó adatokat kell vizsgálni.

Másodlagos jövőbeli terv, egy olyan asztali- vagy mobil alkalmazás fejlesztése, amely segít a rendszer tesztelésében anélkül, hogy azt minden lépés után egy valós járműhöz kéne vinni. Vagyis a programnak képesnek kell lennie egy kommunikációs protokoll felépítésére, valamint ezen keresztül, néhány, a felhasználó által beírt adat továbbítására, amelyek a mikróvezérlős rendszert, hasonló működésre bírják mint amikor a jármű közelében van.

Következő fejlesztési lehetőséggént említeném meg, egy olyan emulátor tervezését, amely egy felhasználói felület segítségével, képes szimulálni ezeket a diagnosztikai elveket.

9. fejezet

Függelék

A diplomadolgozatom mellé csatoltam egy CD-n a műszerfalam mikróvezérlőjét irányító szoftver forráskódját (Bogdan_Norbert_OBD_interface mappában), illetve a dokumentációt pdf formátumban.

Irodalomjegyzék

- [1] Dr. Nagyszokolyai Iván és Dr.Lakatos István. *Gépjármű diagnosztika*. Typotex, Budapest, 2011.
- [2] Dr. Nagyszokolyai Iván és Dr.Lakatos István. *Gépjármű-környezetvédelmi technika és diagnosztika I.* Minerva-Sop – NOVADAT, Győr, 1997.
- [3] Aradi Szilárd. *Fedélzeti diagnosztikai protokollok*. BME, Budapest, 2013.
- [4] Wikipedia. On-board diagnostics. https://en.wikipedia.org/wiki/On-board_diagnostics.
- [5] Budavill online. A csatlakozó lábkiosztása. <http://budavill.hu/pages/A-csatlakoz%C3%B3-1%C3%A1bkioszt%C3%A1sa.html>.
- [6] Dr. Fodor Dénes és Dr.Szalay Zsolt. *Autóipari kommunikációs rendszerek*. Pannon Egyetem, Veszprém, 2014.
- [7] Dr. Fodor Dénes. *Autóipari kommunikációs protokollok – a CAN*. Pannon Egyetem, Veszprém, 2012.
- [8] Debnár Péter. *Soros kommunikáció a gépjárművekben – a CAN*. Budapesti műszaki főiskola, Budapest, 2008.
- [9] Elm Electronics. Elm327 datasheet. <http://elmelectronics.com/DSheets/ELM327DS.pdf>.
- [10] Wikipedia. Mikróvezérlő. <https://hu.wikipedia.org/wiki/Mikrovez%C3%A9rl%C3%B3>.
- [11] Dr. Csernáth Géza. *Mikróvezérlős Rendszerek*. Sapientia Erdélyi Magyar Tudományegyetem, Marosvásárhely, 2014.
- [12] Dr. Halmai Attila. *Szenzor- és aktuátorteknika*. Edutus Főiskola, Budapest, 2011.
- [13] Wikipedia. Uln2003a. <https://en.wikipedia.org/wiki/ULN2003A>.
- [14] Linotux. Hc-03/05 embedded bluetooth serial communication module at command set. http://www.linotux.ch/arduino/HC-0305_serial_module_AT_command_set_201104_revised.pdf.
- [15] Wikipedia. Hall-effektus. <https://hu.wikipedia.org/wiki/Hall-effektus>.
- [16] Microchip. Pic18f4620 datasheet. <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>.
- [17] Wikipedia. Pid codes. https://en.wikipedia.org/wiki/OBD-II_PIDs.

**FUNDAȚIA SAPIENTIA - UNIVERSITATEA SAPIENTIA din CLUJ-NAPOCA
FACULTATEA de ȘTIINȚE TEHNICE și UMANISTE din TÂRGU-MUREŞ
SPECIALIZAREA: Calculatoare**

DECLARAȚIE,

Subsemnatul,

.....
student la specializarea

.....
Facultatea de ȘTIINȚE TEHNICE și UMANISTE din TÂRGU-MUREŞ, de la FUNDAȚIA SAPIENTIA - UNIVERSITATEA SAPIENTIA din CLUJ-NAPOCA, certific că am luat la cunoștință de cele prezentate mai jos și îmi asum, în acest context, originalitatea lucrării mele de diplomă/licență/disertație cu:

titlul.....

.....
coordonator

.....
prezentată în sesiunea ...iunie
2016.....

La elaborarea lucrării de diplomă/licență/disertație, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimelele și referința precisă;
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări dacă nu se indică sursa bibliografică;
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alții autori fără menționarea corectă a acestor surse;
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Data:.....

Semnătura.....

Notă: Se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării;
- indicarea în text a reformulării unei idei, opinii sau teorii și corespondentul în lista de referințe a sursei originale de la care s-a făcut preluarea;
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele etc.;
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim acceptată.

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREŞ
SPECIALIZAREA CALCULATOARE**

Vizat decan

Ş. I. .Dr. ing Kelemen András

Vizat şef catedră

Dr. ing. Domokos József