Állapotgépes járműirányító algoritmus és távolság alapú referencia rendszer implementálása energiahatékony villamos hajtású kísérleti járműben

Írta:
Ország András

III. éves villamosmérnök BSc hallgató - Automatizálási szakirány

Konzulensek:
Szeli Zoltán, kutató, egyetemi tanársegéd
Gulyás Péter, okleveles villamosmérnök

Széchenyi István Egyetem

Győr, 2022

Absztrakt

A TMDK dolgozatom témája egy energiahatékony villamos hajtású kísérleti versenyjármű irányítására szolgáló mikrovezérlőjének szoftveres fejlesztése. Ez a jármű a Széchenyi István Egyetemen működő SZEnergy Team elektromos járművek építésével és fejlesztésével foglalkozó csapat által az évente megrendezésre kerülő Shell Eco-Marathon energiahatékonysági versenyre épített elektromos autó. A dolgozat egyik fő részét képezi egy állapotgéppel felépített biztonságkritikus járműirányító algoritmus, amely lehetővé teszi az irányítás részleges átadását egy felsőbb szintű vezérlés számára, miközben lehetőséget ad a versenyautó pilótájának a vezérlés manuális felülírásához. Ez az állapotgépes rendszer a fent említett verseny 2022-es sikeresen zárt szezonján is jól szerepelt. A dolgozat másik sarkalatos pontja a távolságalapú referenciarendszer. Ennek feladata, hogy külön a versenypályára előzetesen, optimalizáló szoftverrel kiszámított hajtásreferenciát átadja a jármű motorvezérlőjének a pályán megtett távolság alapján. Ezáltal a jármű, akkor is képes az optimalizált referenciát követni, hogyha az autó a felsőszintű irányítás nélkül üzemel. Mivel a fogyasztás csökkentése a verseny szempontjából kulcsfontosságú, ezért a dolgozatomban kitérést teszek a mikrovezérlő energiatakarékos működtetésének megvalósításának módszereiről is.

Tartalomjegyzék

1. Bevezetés

1.1. Shell Eco-Marathon

A Shell Eco-Marathon [1] Európa legnagyobb energiahatékonysági versenye, amely 1985 óta kerül évente megrendezésre. A versenyen kizárólag oktatási intézmények tanulói csapatai vesznek részt, hogy a lehető legjobb energiamérleggel végezzenek az általuk épített versenyjárművükkel, vagyis bemutassák, hogy az ő autójuk tudja egységnyi energiával a legnagyobb távot megtenni. A győri SZEnergy Team 2005 óta vesz részt a Shell Eco-Marathon-on. Aktuális elektromos versenyjárművük a SZEmission, amellyel a Urban Concept kategóriában indulnak, melynek célja, hogy a városi közlekedést szimulálja többek között a verseny során történő kötelező megállásokkal és a járművet közúti közlekedésre alkalmassá tevő kiegészítőkkel. A Battery Electric alkategóriában akkumulátor táplálású elektromos hajtású járművekkel indulnak, ahol 1 kWh energiára számítva kell a legnagyobb távot elérni.

1. ábra SZEmission a SZEnergy Team versenyjárműve [A1]

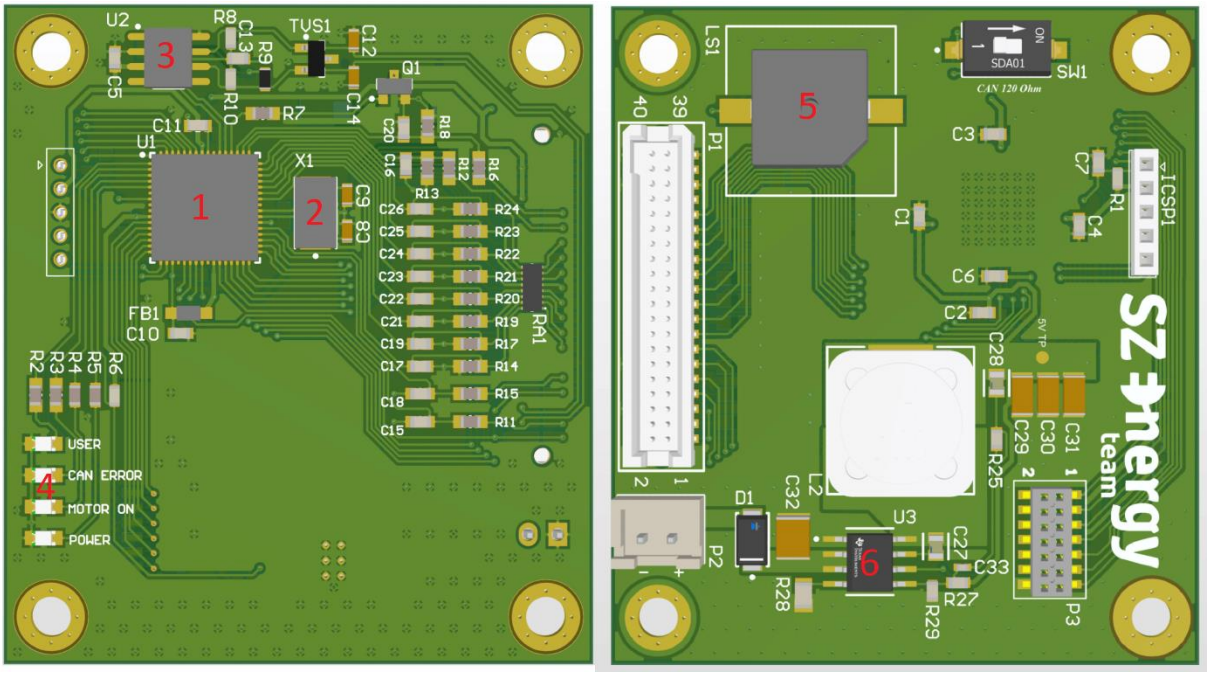
1.2. Célkitűzés

A legfőbb célom a SZEmisson versenyjármű irányítására szolgáló egységének szoftveres felkészítése volt a fent említett versenyre, hogy az el tudja látni a feladatát felsőszintű irányítás hiányában is. Ennek megvalósításához egy biztonságkritikus állapotgépes rendszer megalkotására volt szükség. Ezen felül a csapat által kifejlesztett optimalizáló eljárásból származó versenypályára illesztett sebességgörbe lekövetésre is megvalósítást kellett találni. Ezenkívül a csapat kitűzte célul a 2022-es versenyre, hogy 1 W alatt legyen a jármű álló helyzetben való energiafogyasztása, amihez én is hozzá szerettem volna járulni az említett egység szoftveres irányból történő fogyasztáscsökkentésével.

2. Állapotgépes járműirányítás

2.1. Járműirányító egység

A versenyjármű irányítására szolgáló egysége (Vehicle Control Unit, továbbiakban VCU) egy külön a versenyre tervezett nyomtatott áramkör (továbbiakban NYÁK), amelynek legfontosabb részét a Microchip által gyártott dsPIC33EV256GM106 mikrovezérlő [2] képezi.



2. ábra Mikrovezérlő köré tervezett nyomtatott áramkör

1- Mikrovezérlő
2- Külső oszcillátor (az órajelet biztosítja a mikrovezérlőnek)
3- CAN transceiver (CAN kommunikációt biztosítja)
4- LED-ek (fényjelzést adnak a pilótának)
5- Buzzer (hangjelzést ad a pilótának)
6- Feszültség regulátor (5V feszültséget biztosít az áramkörnek)

A mikrovezérlő egy egyetlen integrált áramköri chipen (IC) található, általában vezérlési feladatokra optimalizált számítási egység. Egy mikroprocesszorból és az áramköri lapkájára integrált perifériákból épül fel.

A mikrovezérlő típus megnevezését kielemezve is már sok tudható meg róla:

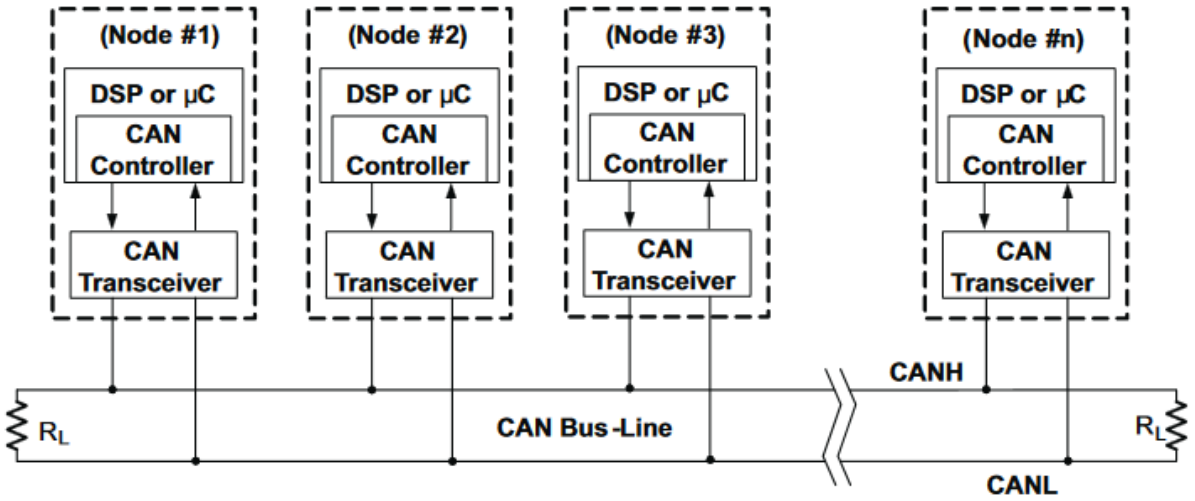
dsPIC33EV256GM106

- dsPIC = Microchip által gyártott digitális jelfeldolgozó vezérlő
- 33 = 16-bites architektúra
- EV = Enhanced Voltage, vagyis 5 Volton üzemel 3,3 Volt helyett
- 256 = 256 KB program memória
- GM1 = Általános felhasználású plusz motor vezérlő család
- 06 = 64 pinnel rendelkezik

Ezen adatok közül a 16-bites architektúra azt jelenti, hogy 16 bites adatbusszal rendelkezik a mikrovezérlő, így kevesebb óraciklus alatt tud elvégezni egy utasítást, mint például egy 8 bites. A 256 KB program memória elegendő helyet ad nagyobb kiterjedésű és komplikáltabb programkódok megvalósítására. A 64 pin pedig megfelelő mennyiségű ki-és bemenetet ad az mikrovezérlő perifériái számára (CAN modul, ADC, külső oszcillátor stb.), valamint a versenypilóta általi bemeneteknek (kapcsolók, nyomógombok).

A mikrovezérlő egyik legfontosabb tulajdonsága, hogy rendelkezik CAN interfésszel, amely kommunikációt biztosít jármű további elektronikai egységeivel. A Controller Area Network (továbbiakban CAN) robusztussága és üzembiztonsága miatt a legelterjedtebb kommunikációs protokoll az autóiparban [3]. Soros kommunikációt, vagyis az adatok bitenként egymás utáni továbbítását biztosítja. A CAN Multi-Master elven működik, azaz a kommunikáció egyik tagja sincs alárendelve a másiknak. Különlegessége, hogy nem a hálózaton lévő két pont közötti adatáramlást teszi lehetővé, hanem egy pont által a hálózatra küldött adatok az összes többi pont számára is elérhetővé válnak. Ennek biztosításához két jelvezeték szükséges a CANH és a CANL (3. ábra), amelyek 120 Ohm hullámimpedanciájú sodrott érpárral vannak megvalósítva. Ahhoz, hogy több egység által egyszerre küldött üzenetek ne okozzanak ütközést, minden üzenet azonosítóval van ellátva, amely egyben megadja az üzenet prioritását is a hálózaton. Minden kommunikációban résztvevő egységnek rendelkeznie kell egy vezérlővel, amely értelmezi a CAN üzeneteket, felügyeli a protokoll betartását, keresi a hibákat és ha szükséges közbeavatkozik, valamint rendelkeznie kell egy illesztővel, amelynek feladata, hogy a vezérlő által küldött üzeneteket kiültesse a fizikai hálózatra, vagy azokat beolvassa onnan, illetve fenntartsa a protokoll működéséhez szükséges

feszültségszintet. A vezérlő szerepét a mikrovezérlő CAN interfésze, az illesztőjét pedig a 2. ábrán látható CAN Tranceiver látja el.



3. ábra CAN hálózat [A2]

Az időzítők (Timers) [4] a mikrovezérlő belső órajeléből vagy egy külső órajelből állítják elő a működésükhöz szükséges frekvenciát. Egy számláló egy megadott értéktől számol le az időzítő frekvenciája szerint. Mikor a számláló nullához ér, újra felveszi a megadott értéket és szükség esetén megszakítást is generál. Ennek használatával elő lehet állítani tetszőleges időközönként megszakításokat, vagy le lehet kérdezni a számláló aktuális értékét, hogy meg lehessen állapítani azt, hogy két esemény közt mennyi idő telt el. Az általam használt időzítő periodikus megszakításokat generál többek között CAN üzenetek küldésére és távolság számolására.

Az Interrupt Controller [5], vagy megszakítás vezérlő az időzítőktől és más perifériáktól érkező program-megszakításokat kezeli. Ezeknek prioritási értékeket ad, amely szerint el tudja dönteni, hogy két egyidejű megszakítás esetén melyiket továbbítsa előbb a mikrovezérlő processzora felé. Továbbá megadja az Interrupt Service Routine-t (ISR), vagyis, hogy adott megszakítás esetén melyik programrészlet fusson le.

A Direct Memory Access (továbbiakban DMA) [6] csatornák direkt hozzáférést adhat egyes perifériáknak a rendszermemóriához, ezzel nem terhelve a mikrovezérlő processzorát és felgyorsítva az adatátviteli folyamatot. Esetemben a CAN protokollhoz használom ezt a lehetőséget. Üzenetek küldésekor egy időzítő megszakítás hatására a DMA átteszi a küldeni kívánt adatot a CAN modul ideiglenes tárhelyére, amely ezután kiküldésre kerül. Üzenet fogadásakor a CAN modul küld megszakítást, hogy megtelt az ideiglenes tárhelye egy fogadni kívánt üzenettel. Erre megszakításra a DMA áthelyezi üzenet adatait a kívánt memóriahelyre.

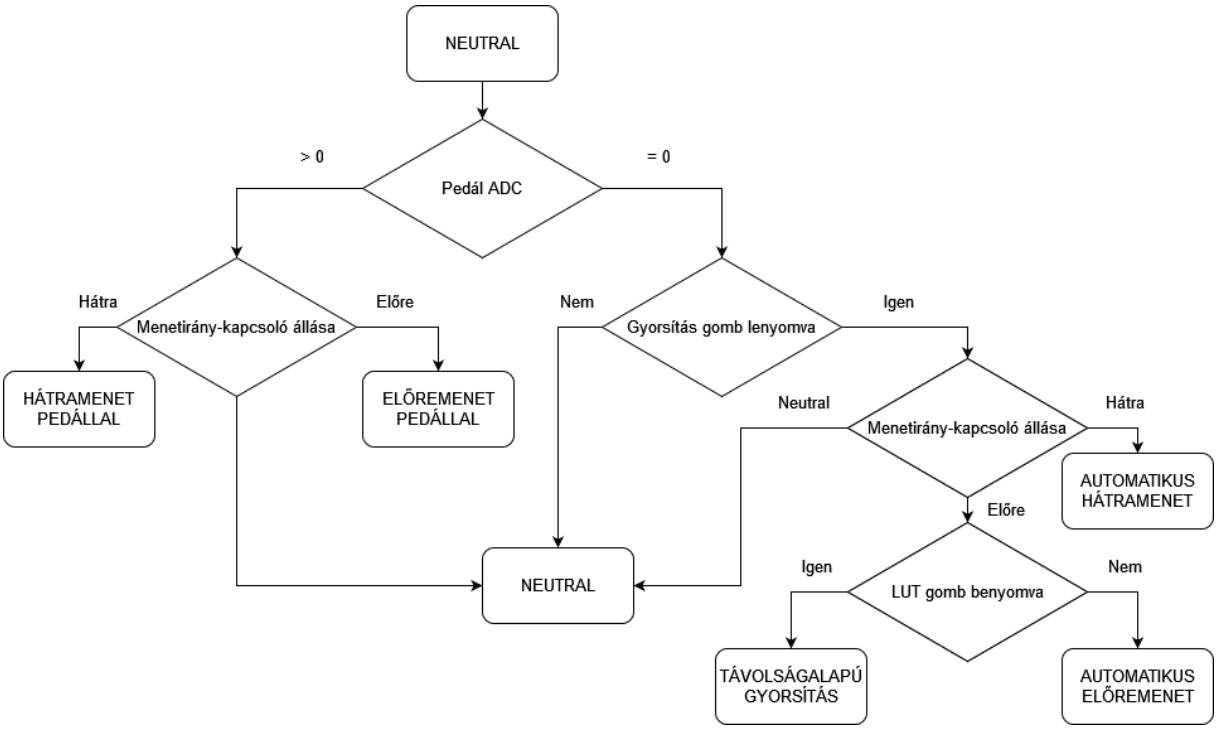
Az Analog Digital Converter (továbbiakban ADC) [7] a bejövő analóg jeleket alakítja át eltárolható digitális jelekké. Az általam használt mikrovezérlő ADC szukcesszív approximációt vagy más néven fokozatos közelítést alkalmaz a digitális jel előállításához. Ekkor a bemeneti analóg jelet összehasonlítja egy ismert referenciajellel. Attól függően, hogy bemeneti jel kisebb vagy nagyobb a referenciajelnél, eltárol egy 0-át vagy egy 1-et, és kivonja az referenciajel értékéből vagy hozzáadja referenciajel értékéhez annak a felét és az lesz az új referenciajel. Ez annyiszor játszódik le ahány bites az ADC felbontása, és az eredményül kapott digitális jel is annyi biten lesz eltárolva. A mikrovezérlő 12 bites ADC-je jelen esetben a gázpedál potenciométer feszültség értékének feldolgozásra van használva.

2.2. Állapotgép

Az állapotgép egy matematikai modell, amit gyakran használnak programfejlesztés, digitális áramkörök tervezése és egyéb mérnöki munka során [8]. Egy időpillanatban mindig csak egy meghatározott állapotban lehet az állapotgép. Egyik állapotból a másikba jól meghatározott feltételek teljesülésével lehet átlépni. Ezek a feltételek a bemeneti jelek és a jelenlegi állapot kombinációjából állhatnak. Ezen felül az állapotgépnek rendelkeznie kell egy kezdő állapottal, amely minden indításkor biztonságos kiindulópontként szolgál. Mivel minden időpillanatban egy előre meghatározott állapotban van a rendszer, így biztonságosabb lesz a működése, mert a végrehajtandó utasítások is ilyen állapotokhoz kötöttek és nem fordulhat elő a kimeneteknek olyan kombinációja, amelyet nem a rendszer alkotója tervezett meg.

2.3. Megvalósítás

Ebben az esetben menetirányok és a hajtások módjának meghatározására készítettem egy állapotgépet, mely a kormányon lévő gombok és kapcsolók, a gázpedál, és egyéb bementek alapján kerül a megfelelő állapotba. Az állapotgép működését egyszerűsítve a 4. ábra szemlélteti.



4. ábra Járműirányító állapotgép egyszerűsített folyamatábrája

Látható, hogy amikor a menetirány-kapcsoló Neutral állásban van, akkor csak a NEUTRAL állapotot veheti fel az állapotgép, ezzel biztosítva a biztonságos működést és a gyors hajtáselvételt. Abban az esetben, ha a menetirány-kapcsoló nem Neutral állásban van és az ADC a meghatározott küszöbértéknél nagyobb feszültségértéket kap a pedál potenciométerétől, a motorvezérlő azzal a feszültségértékkel arányosan kap hajtásreferenciát. A LUT gomb egy a kormányon megtalálható gombok egyike, amely a távolságalapú referenciás hajtásra lett kijelölve a mikrovezérlő programon belül. A programban az állapotgép logikai megvalósítása optimalizálási okok miatt eltér a fenti ábráétól, de kimenetelében ugyanazt adja. Az állapotgép kódját az 1. mellékelt tartalmazza.

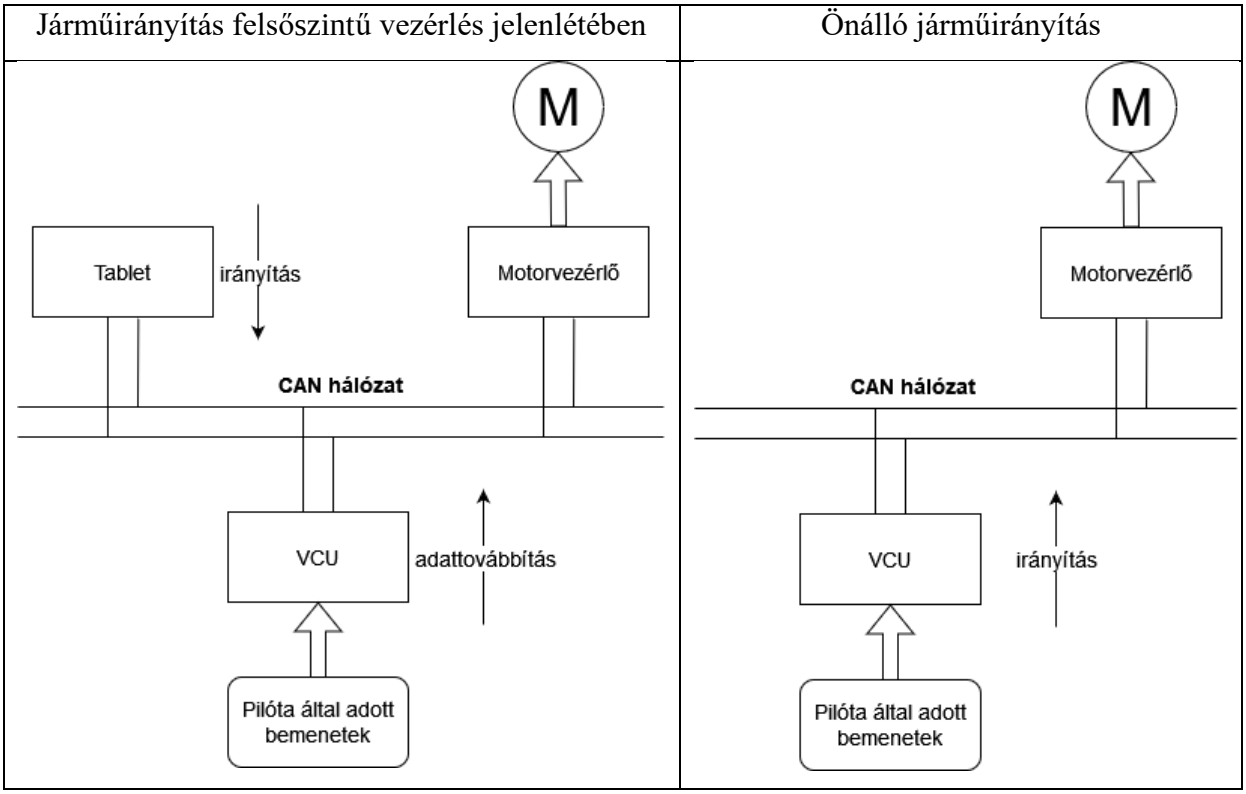
A függvény végén el kell tárolni az előző állapotot, mivel amikor valamilyen hajtási állapotból a NEUTRAL állapotba lép át a programunk, akkor el kell altatni a motorvezérlőt, hogy addig az minimális fogyasztással üzemeljen. Ezt úgy lehet megtenni, ha az állapotváltás után egy darab 0-s hajtásreferenciát küld a VCU a motorvezérlőnek CAN-en keresztül. Az aktuális állapot szerinti irányítás a program főciklusában valósul meg, amely a 2. mellékletben látható, míg az állapotgép átmeneti táblázata (1. táblázat) alább látható.

Jelenlegi állapot	Következő állapot	Feltételek			
		Pedál ADC	Menetirány-kapcsoló állása	Gyorsítás gomb lenyomva	LUT gomb benyomva
X	NEUTRAL	X	Neutral	X	X
NEUTRAL	ELŐREMENTET PEDÁLLAL	> 0	Előre	X	X
NEUTRAL	HÁTRAMENET PEDÁLLAL	> 0	Hátra	X	X
NEUTRAL	AUTOMATIKUS ELŐREMENET	= 0	Előre	Igen	Nem
NEUTRAL	AUTOMATIKUS HÁTRAMENET	= 0	Hátra	Igen	X
NEUTRAL	TÁVOLSÁGALAPÚ GYORSÍTÁS	= 0	Előre	Igen	Igen

1. táblázat Állapotátmenetek

Észrevehető, hogy mindegyik állapotból közvetlenül át lehet lépni a NEUTRAL állapotba, és minden egyéb állapotba csak ezen keresztül lehet eljutni, amely tovább bizonyítja, hogy a rendszer biztonságkritikus működést helyezi előnybe.

Amennyiben van lehetőség felsőszintű irányítási jelek alkalmazására, akkor nem a mikrovezérlő dönt a járműirányítást befolyásoló referencia jelekről, ilyenkor csak a bemeneti értékek továbbításáért felelős. Ez az eset a fent említett mellékletben (2. melléklet) szereplő kódrészlet elején vannak lekezelve. Ennek és a VCU-val való önálló irányításnak az esetét a 5. ábra szemlélteti.



5. ábra Járműirányítás két esete

A felsőszintű irányítás szerepét egy táblaszámítógép biztosítja, melyen egy LabVIEW-ban írt alkalmazás fut. A táblaszámítógép egy CAN feldolgozóegységen keresztül csatlakozik a kommunikációs hálózathoz. A felsőszintű irányítás jelenlétében a VCU csak továbbítja a bemeneti jeleket, amelyeket a táblaszámítógép feldolgoz és azok alapján küld irányító jelet a motorvezérlőnek. Abban az esetben, ha nem áll rendelkezésre a felsőszintű vezérlés, a VCU-nak kell az adatokat feldolgoznia és irányító jelet küldenie a motorvezérlőnek.

Korábbi szoftver változatokban az automatikus gyorsítás és lassítás a pillanatnyi sebesség függvényében egyenlet szerint történt, de tesztelések során kiderült, hogy érdemesebb gyorsításhoz mindig 100%-os hajtásreferenciát használni. Viszont, hogy ez ne egyik pillanatról a másikra történjen, és meg legyen kímélve a hajtáslánc a túl hirtelen erőhatásváltozásoktól, limitálni kell a hajtásreferencia értékének változását. Ezt a feladatot a Rate Limiter látja el, ami az alábbi kódrészletben látható:

```
549  void RateLimiter(void){
550    if(2000 + conversion.ADC_Word - prev_ref_conversion.ADC_Word >= 2000 +
551  RATE_LIMIT){
552        conversion.ADC_Word = prev_ref_conversion.ADC_Word + RATE_LIMIT;
553    }
554    prev_ref_conversion.ADC_Word = conversion.ADC_Word;
555  }
```

A RATE_LIMIT változóval lehet meghatározni, hogy milyen gyorsan menjen végbe a hajtásreferencia változás, vagyis a gyorsítás.

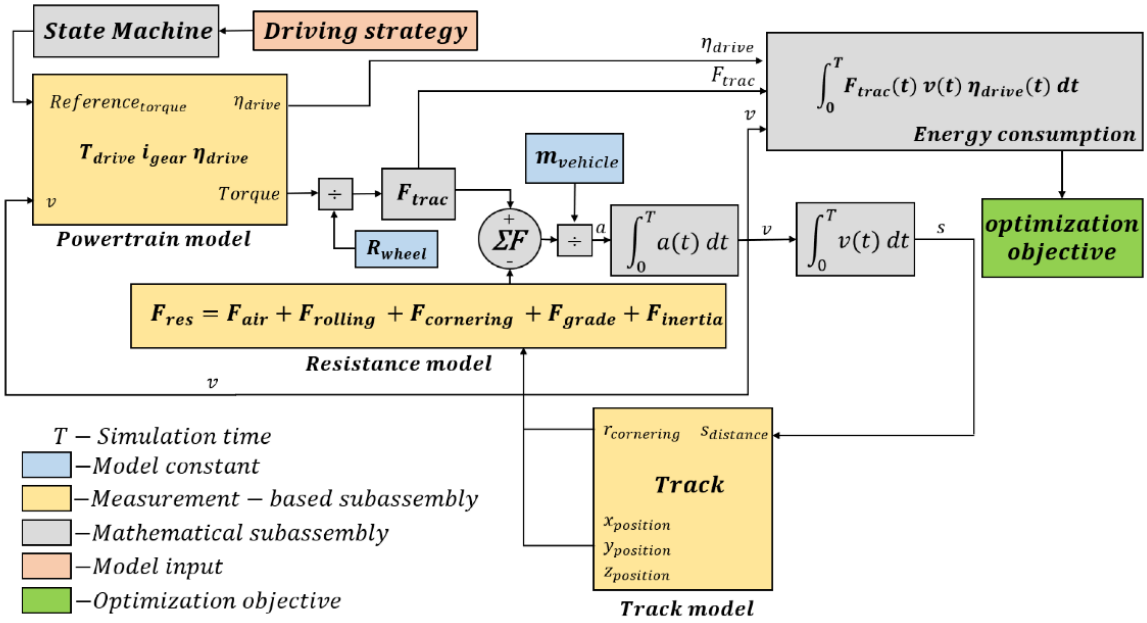
3. Távolságalapú referenciarendszer

A távolságalapú referenciarendszer lényege, hogy a járműhajtáslánc legjobb fogyasztásának eléréséhez egy előre meghatározott optimális sebességgörbét kövessen a versenyjármű. Ezt azzal valósítottam meg, hogy a motorvezérlőnek a versenypályán megtett pillanatnyi megtett távolság alapján ad hajtásreferenciákat a VCU. Az optimális sebességgörbe az adott versenypályára van szabva, a jármű hajtáslánckonfigurációját figyelembe véve.

3.1. Optimalizációs eljárás

Ez a rész röviden bemutatja, az eljárást, amely eredményeképp kapjuk meg a fent említett optimális sebességgörbét, amellyel mi dolgozunk. Ezt bővebben az erről írt tanulmány mutatja be [9].

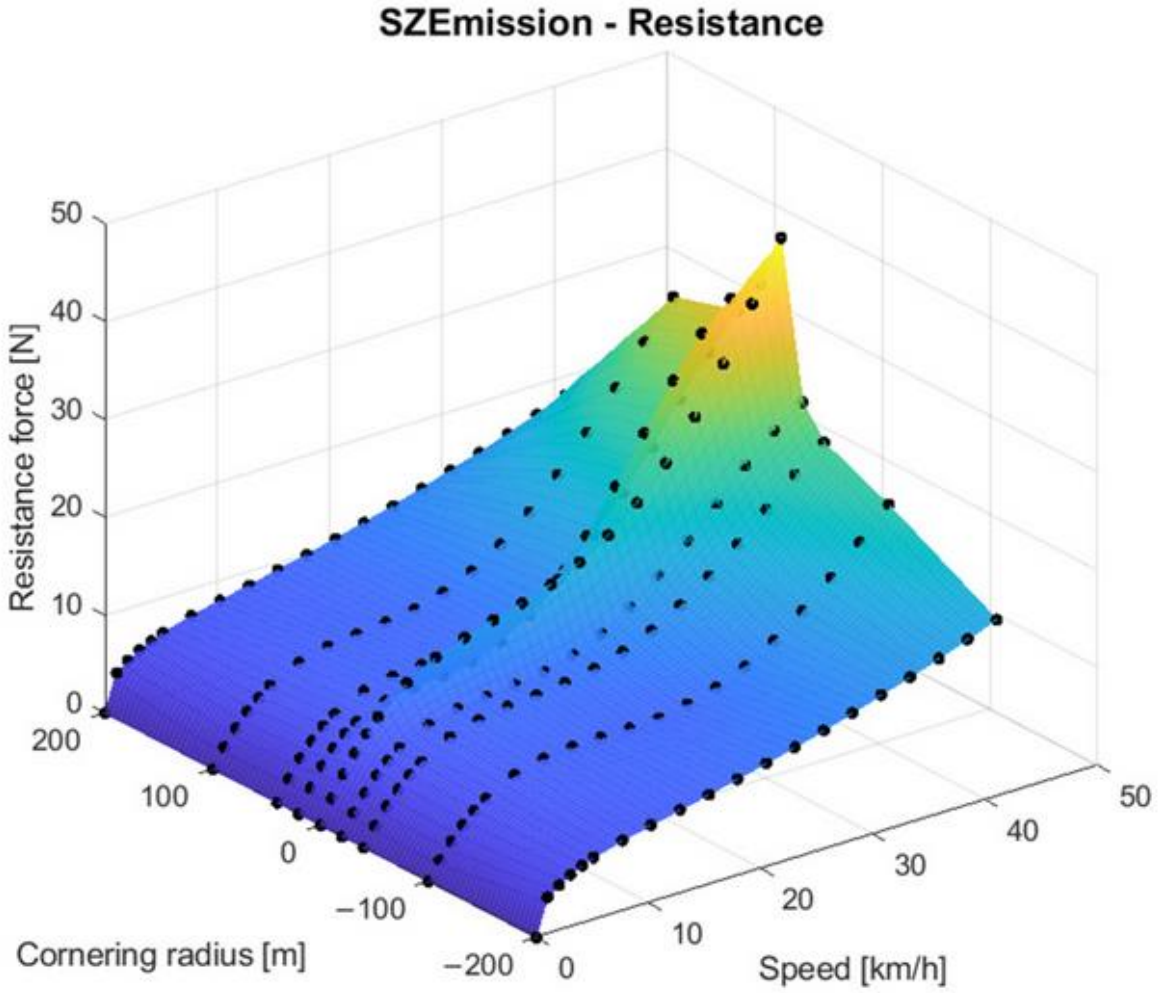
Az optimalizáció eljáráshoz egy mérésalapú matematikai járműmodellt (6. ábra) alkotott meg a SZEnergy Team, amely magában foglalja a hajtáslánc, a jármű ellenállás, valamint az adott versenypálya mérések által megkapott modelljeit.



6. ábra Mérésen alapuló járműmodell vázlata [A3]

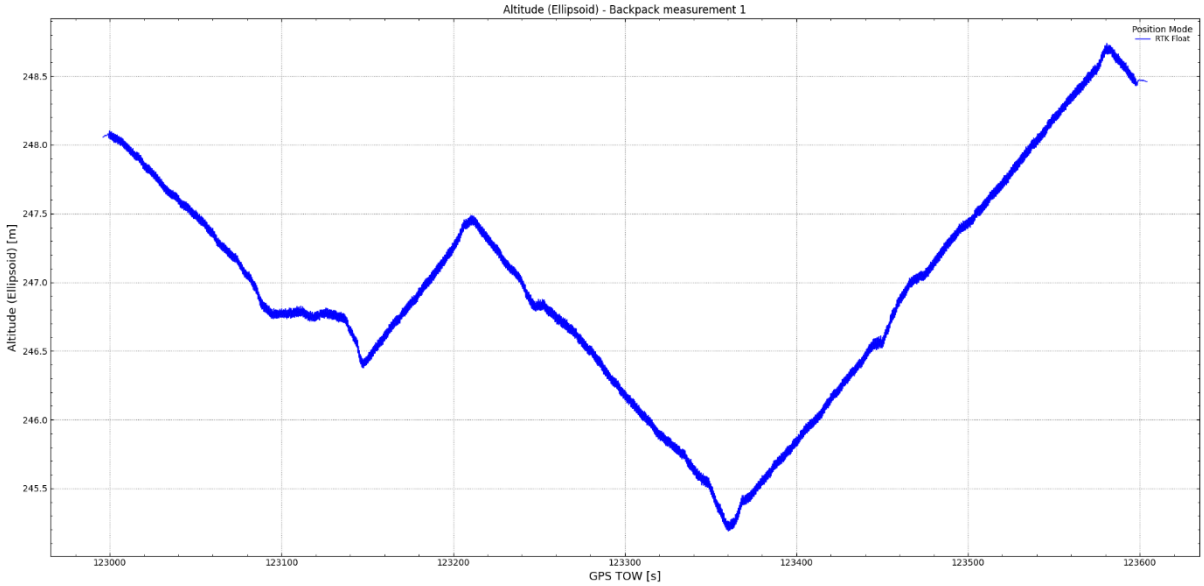
A hajtáslánc modell tartalmazza az adott villanymotor és áttétel tesztpadon mért eredményeiből megkapott sebesség-nyomaték mezőt, amely megadja a komplett hajtáslánc hatásfokát az említett változók alapján.

Az ellenállás modell tartalmaz minden olyan erőhatást, amely befolyásolja a jármű mozgását. A légellenállás és gördülési ellenállás a jármű alakjától, míg az inerciális ellenállás a jármű tömegétől függenek. Az emelkedésből és a kanyarodásból származó ellenállások a pálya paramétereinek a függvényében változnak, ezért van szükség rá szükség, hogy a teljes járműmodell tartalmazz a versenypálya modelljét is. Az ellenállás modell (7. ábra) az egyenes gurulás és kanyarodás során mért ellenállásértékek kombinációjából született meg.



7. ábra A SZEmission háromdimenziós ellenállás modellje [A4]

A versenypálya modell az adott pálya földrajzi koordinátáiból kapható meg. Ezeket az értékeket vagy a pályáról elérhető publikus földrajzi adatokból, vagy a pálya egyedi felmérését elvégezve egy a járműbe szerelhető RTK pontosított GNSS rendszerből lehet kinyerni. Egy a járműbe szerelhető GNSS mérés eredményének részlete látható az alábbi 8. ábra.



8. ábra RTK pontosított magassági adatok [A5]

A RTK pontosított mérések segítségével 10 cm-es pontossággal határozható meg az adott versenypálya geográfiai felépítése, mely lehetőséget biztosít a lejtő és emelkedő szakaszok pontos figyelembevételére az optimális hajtási stratégiához.

Optimalizálás során a járműmodell egyes paramétereit a MATHLAB genetikai algoritmusa változtatja nemdeterminisztikus eljárásokkal, mint például mutáció, szelekció. Ezzel ki lehet választani a legoptimálisabb hajtásláncot vagy vezetési stratégiát egy adott versenypályához.

3.2. Távolság meghatározása

A jármű pillanatnyi távolságát a pályán a jármű sebességéből és az eltelt időből lehet megállapítani, sebességet pedig a kerék fordulatszámából. A távolság kiszámítása a következőkben leírt módon történik.

A jobb hátsó keréken lévő enkóder egység számolja a kerék fordulatszámát és a számolt értéket kiküldi a CAN hálózatra 20 Hz-es (50 ms-onként) gyakorisággal. A VCU mikrovezérlője kiolvassa ezt az értéket és a következő képlettel (1) kiszámítja a jármű sebességét.

$$v = n \cdot \frac{\pi \cdot d}{60} \tag{1}$$

v: jármű sebessége [m/s]

n: kerék fordulatszáma [1/min]

d: kerék átmérője [m]

Mivel 50 ms-onként érkezik új fordulatszám érték, ezért nem szükséges 50 ms-nál kisebb időközönként új távolságot számolni. Ezt úgy valósítottam meg, hogy amikor a progmramban lévő számláló 50 ms leteltével megszakítja a program futását, az akkor számított sebességgel kiszámítom az 50 ms alatt megtett távot, és ezt az értéket hozzáadom az összesen megtett távolsághoz. Ez az alábbi képlet (2) szerint történik.

$$s = \sum_{T=0}^{n} \frac{v \cdot 100}{20} \tag{2}$$

s: megtett távolság [cm]

T: 50 ms-os diszkrét időköz

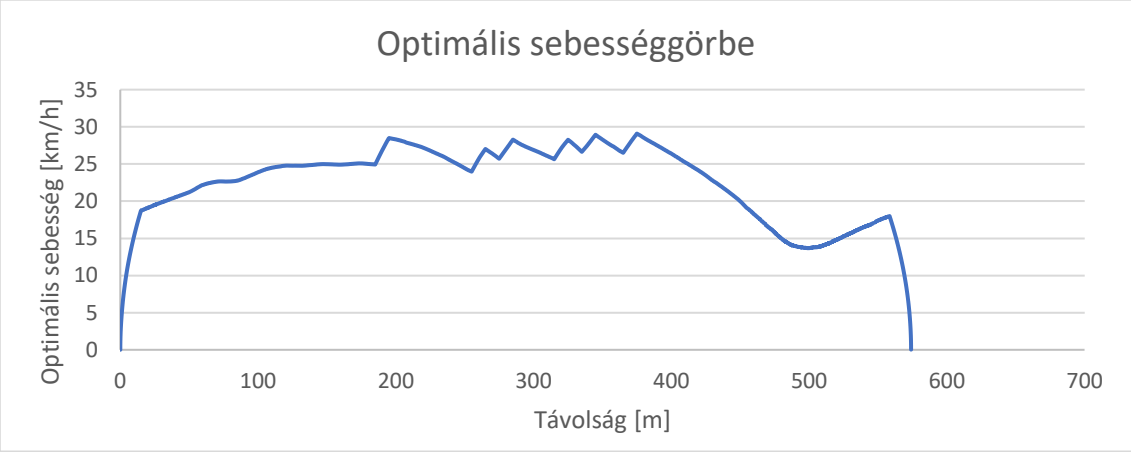
n: az eddig eltelt T időközök száma

v: jármű sebessége [m/s]

A távolság számolása minden a versenypályán való új kör megkezdésével újraindul amikor a pilóta megnyomja az új kör gombot, aminek hatására a megtett távolságot lenullázom. Ezért, ha van némi eltérés a számított és a valós távolság között, ez nem fog összeadódni minden körben.

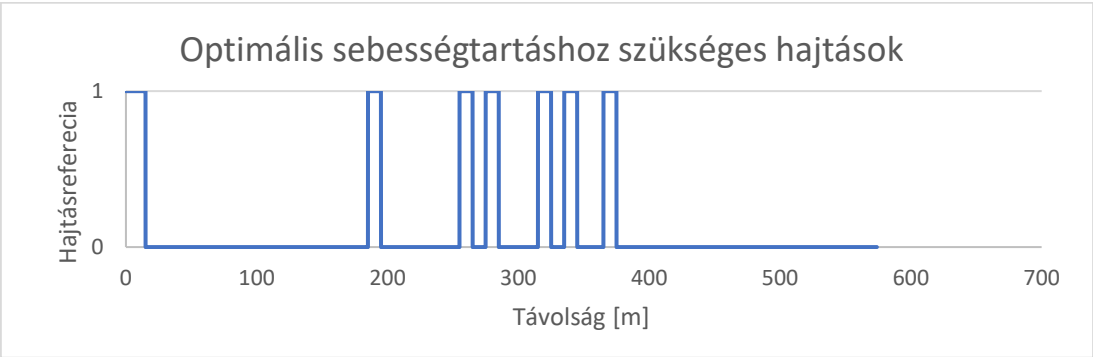
3.3. Távolsághoz hajtásreferencia hozzárendelése

Adott versenypálya felmérése után az arra a pályára vonatkozó optimális sebességgörbe kerül kiszámításra az optimalizáló programmal (9. ábra). Ezt a görbét kell lekövetni, hogy a lehető legalacsonyabb fogyasztással lehessen végig menni a pályán. Ehhez a megfelelő nyomatékreferenciát kell küldeni a CAN hálózaton keresztül a motorvezérlőnek. A nyomatékreferencia 10 biten van meghatározva, ahol 0-ás értéknél nincs hajtás 1023-mas értéknél pedig 100%-os hajtás van.



9. ábra: Pályára vonatkozó optimális sebességgörbe

A tesztelések során kiderült, hogy érdemesebb csak 100%-os gyorsítással hajtani, mert ilyenkor az idő nagyobb részében nem kell hajtani és ki lehet kapcsolni a motorvezérlőt, ezzel tovább csökkentve a jármű fogyasztását. Mivel 1024 hajtásérték helyett csak kettő közül kell választani egy adott távolságban, ezért elegendő 1 biten bool formátumban eltárolni ezeket, így több adatot tudunk tárolni a mikrokontroller memóriájában, vagyis nagyobb felbontásban lehet az érték hozzárendelést megvalósítani, mint korábban. A 10. ábra az így megvalósított sebességgörbe lekövetéséhez szükséges hajtásreferenciákat mutatja.



10. ábra Távolság szerinti optimális hajtások

Észrevehető az előbbi két ábrát összehasonlítva, hogy 500 és 550 méter között nagyobb sebességre tesz szert a jármű anélkül, hogy gyorsítana. Ez az adott pálya szintkülönbségének köszönhető, mivel az utolsó szakaszon lejtőn megy a jármű. A verseny során minden kör végén meg kell állni, ennek köszönhetően az első körben is ugyanazt a gyorsításprofilt lehet használni, mint az utolsóban.

Az optimalizáló szoftver századmásodperc pontossággal az eltelt időhöz kötve adja meg a számított hajtásreferenciákat, mivel nekem távolsághoz kötve kell ezen értékeket megadnom, ezért az adattáblázat átalakítására van szükség. Ezt az 2. táblázat szemlélteti.

idő [s]	távolság [m]	sebesség [km/h]	hajtásreferencia
5.72	14.544617	18.44561	1
5.73	14.5959	18.478096	1
5.74	14.647273	18.510559	1
5.75	14.698736	18.542997	1
5.76	14.750289	18.575421	1
5.77	14.801933	18.607817	1
5.78	14.853666	18.640149	1
5.79	14.905489	18.672415	1
5.8	14.957402	18.704616	1
5.81	15.009404	18.736791	0
5.82	15.061455	18.740325	0
5.83	15.113517	18.743923	0
5.84	15.165588	18.747586	0
5.85	15.21767	18.751309	0
5.86	15.269762	18.755048	0
5.87	15.321865	18.758809	0
5.88	15.373978	18.762601	0
5.89	15.426101	18.766424	0

→

távolság [m]	hajtásreferencia
14.5	1
14.55	1
14.6	1
14.65	1
14.7	1
14.75	1
14.8	1
14.85	1
14.9	1
14.95	1
15	0
15.05	0
15.1	0
15.15	0
15.2	0
15.25	0
15.3	0
15.35	0

2. táblázat Optimalizált adatok átalakítása

Felmerülhet, hogy nagyobb sebességeken nagyobb távolságfelbontással adom meg a hajtásreferenciákat, mint az optimalizáló program. Erre a redundanciára azért van szükség, mert kisebb sebességeken az optimalizáló program már sokkal kisebb távolságközökkel adja meg a kellő hajtásértékeket és ezt a programnak minél jobban le kell tudnia követni. Ahhoz, hogy hozzá lehessen férni programfutás közben ezekhez a hajtásértékekhez, LookUp Table-t használok.

A LookUp Table (LUT) egy számítástechnikában használt adatstruktúra, mellyel csökkenthető a futásidőben végzendő számítási műveletek száma a programmemória foglalásáért cserébe. Ezt úgy éri el, hogy előre kiszámított értékeket tárol és rendel hozzá meglévő értékekhez. Ezt a hozzárendelést direkt módon végzi el, anélkül, hogy a tárolt elemeken végig kellene mennie. Az előre kiszámított értékeket az eszköz programozásakor égetem be a mikrovezérlő memóriájába, melyet később könnyedén használok a meghívott ciklusokban. Minél nagyobb tárhely áll rendelkezésre a memóriában, annál nagyobb távolságfelbontásban lehet az hajtásértékeket megadni.

Esetemben a LUT-t egy egyszerű tömbbel valósítottam meg, ahol az optimalizált hajtási referenciák a tömb értékei. Ennek a tömbnek a feltöltésére, vagyis ahhoz, hogy az Excel táblázatból egy C program tömbjébe egyszerűen átmásolható legyen, egy segéd C programot (3. melléklet) írtam. A hozzárendeléshez a távolságértékeket egész számú indexekké alakítjuk át a következő képlet (3) segítségével.

$$i = \frac{s + s_{step}}{s_{res}} \tag{3}$$

i: kiszámolt index, ami alapján kapjuk meg a hozzárendelt hajtásreferenciát

s: megtett távolság [cm]

s_{step}: távolság eltolás [cm], ha szükséges akkor ezzel lehet előbbi értéket megkapni, ezzel kompenzálva a felmerülő késéseket

s_{res}: a távolság felbontása, ez esetben 5 [cm]

Mivel nem egyszerű osztást alkalmazok, hanem a C nyelvben használt ”/” operátort, ezért a kapott index már egész számra lesz lefelé kerekítve. Ha az indexen tárolt érték 1, akkor a VCU 100%-os hajtásreferenciát küld, ha 0 akkor pedig nincs hajtás, vagyis 0-ás lesz küldve a motorvezérlőnek. Ez a 4. melléklet kódrészletében látható.

4. Fogyasztáscsökkentési megoldások

Mivel a verseny során nem csak a hajtáslánc fogyasztását mérik, hanem a járműben lévő szinte minden elektronikai elemének fogyasztását, emiatt a mikrovezérlő és az azt körülvevő nyomtatott áramkör energiafogyasztásának csökkentésére is szükség van. Ezt a következő megoldásokkal értem el.

4.1. Mikrovezérlő altatása

A mikrovezérlőaltatás az egyik legelterjedtebb módja a fogyasztáscsökkentésnek, erre a gyártók különböző altatási módokat tesznek elérhetővé [10]. Az általam használt mikrovezérlőnél 11. ábra szerinti lehetőségek közül lehet választani.

Mode	CPU Clock (Fcy)	System Clock (Fosc)	Peripheral Clock (Fp)	All Memory Retained
Doze	Slower	Full Speed	Full Speed	Yes
Idle	Stopped	Full Speed	Full Speed	Yes
Sleep	Stopped	Stopped	Stopped	Yes
Low Voltage Sleep	Stopped	Stopped	Stopped	Yes

11. ábra Mikrovezérlő altatási módok [A6]

A Sleep Mode-ot esetemben nem lehet használni, mivel szükség van rá, hogy az időzítő megszakítással fel tudja ébreszteni a mikrovezérlőt ebből az állapotból. Az időzítő pedig a rendszer órajelből (Fosc) számítja a saját órajelét, ami ebben a módban meg van állítva. Ennek elkerülése végett a következő legtakarékosabb módot az Idle Mode-ot választottam. Minden főciklus lefutása végén altatom el a mikrovezérlőt, amit majd az időzítő vagy egyéb perifériák fognak felkelteni.

4.2. Órajel csökkentése

Az mikrovezérlő órajelének, vagyis a rendszer frekvenciájának csökkentése a másik nagyon elterjedt módja a fogyasztáscsökkentésnek. Ilyenkor viszont figyelni kell arra, hogy csak annyira legyen az órajel lecsökkentve, hogy az a mikrovezérlő működését ne befolyásolja. A

főciklus le tudjon futni időzítő megszakítások között, a perifériák, amik az órajelüket a rendszer órajelből származtatják megfelelő módon működjenek. Alapesetben nem szokták ezt a módszert együtt használni az altatással, azonban most csak Idle Mode-ba lép a mikrovezérlő altatáskor és ilyenkor a perifériák is fogyasztanak. Miután megmértem különböző konfigurációkban az energiafogyasztást, bebizonyosodott, hogy érdemes a két módszert együtt használni. Azonban stabil működés megőrzése érdekében 80 MHz-ről csak 60 MHz-re csökkentettem a rendszerfrekvenciát, így hagyva egy kis biztonsági zónát a szélsőséges üzemelés közbeni esetekre.

4.3. Egyéb alkalmazott fogyasztáscsökkentési megoldások

A két fent említett módszeren kívül máshogyan is hozzá lehet járulni a fogyasztás csökkentéséhez.

Ezek közül az egyik, hogy az eltárolt bemenetek megváltozásai nem meghatározott időközönként kerülnek frissítésre, hanem csak, amikor változás történik az értékükben. Ilyenkor a változás egy megszakítást generál és csak ekkor frissülnek az eltárolt bemeneteket. Ezt használva csak akkor ébred fel a mikrovezérlő, mikor tényleg szükséges [11].

A mikrovezérlő nyomtatott áramköri lapján lévő LED-ek alapesetben lekapcsoltam, és csak hibajelzésre vannak használva. Mivel ezek a LED-ek aszerint lettek kiválasztva, hogy a pilótának tudjanak jelzést adni vezetés közben, ezért a fogyasztásuk a mikrovezérlőjéhez képest magasnak számít. Az említtet fogyasztáscsökkentési megoldás együttes alkalmazásával sikerült a VCU fogyasztását, eddigi értékének kevesebb mint felére lecsökkenteni.

5. Összegzés

A célom a SZEnergy Team által megalkotott SZEmission nevű energiahatékony versenyjármű irányítására szolgáló egység felkészítése volt, hogy a 2022-es Shell Eco-Marathon-t eddigieknél is jobb energiamérleggel tudja zárni a csapat, hogy legalább az öt legjobban teljesítő csapat között legyen, akár felsőszintű irányítás nélkül is. Ennek megvalósításához az egység mikrovezérlőének szoftveres fejlesztésére volt szükség, aminek fő eredménye egy járműirányításra szolgáló állapotgép rendszer volt, mely helyt tudott állni a verseny során. Egy távolságalapú referencia rendszer létrehozása is sor került, amely bár képes kellő pontossággal a megtett távolságot kiszámítani és ez alapján a megfelelő hajtásreferenciát visszaadni, ahhoz, hogy versenyen is alkalmazható legyen még számos tesztnek kell alávetni. Ezenfelül a fogyasztáscsökkentési módszerek együttes alkalmazásával sikerült az egység eddigi 428 mW-os fogyasztását 178 mW-ra csökkenteni.

Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulenseimnek Gulyás Pétérnek és Szeli Zoltánnak,
valamint a SZEnergy Team-nek TMDK dolgozatom írása alatt nyújtott szakmai segítségükért.

A kutatás a TKP2021-NKTA-48 számú projekt keretében az Innovációs és Technológiai Minisztérium Nemzeti Kutatási, Fejlesztési és Innovációs Alapból nyújtott támogatásával, a TKP2021-NKTA pályázati program finanszírozásában valósult meg.

Irodalomjegyzék

[1] Shell Eco-Marathon:

https://www.shell.hu/about-us/tarsadalmi-felelossegvallalas/shell-eco-marathon-europe.html

utolsó látogatás: 2022. november 4.

[2] Microchip által gyártott dsPIC33EV256GM106 mikrovezérlő adatlapja:

https://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33EVXXXGM00X-10X-Family-Data-Sheet-DS70005144H.pdf

utolsó látogatás: 2022. november 4.

[3] Controller Area Network:

https://www.ti.com/lit/an/sloa101b/sloa101b.pdf

utolsó látogatás: 2022. november 4.

[4] Időzítők:

www.microchip.com/DS70362

utolsó látogatás: 2022. november 4.

[5] Megszakítások:

www.microchip.com/DS70000600

utolsó látogatás: 2022. november 4.

[6] Direct Memory Access:

www.microchip.com/DS70348

utolsó látogatás: 2022. november 4.

[7] Analog Digital Converter:

www.microchip.com/DS70621

utolsó látogatás: 2022. november 4.

[8] Peng Zhang., *Industrial Control Technology*, 2006

[9] Tanulmány az optimalizáló eljárásról:

Pusztai Z, Kőrös P, Szauter F, Friedler F.: *Vehicle Model-Based Driving Strategy Optimization for Lightweight Vehicle. Energies*. 2022, 15, 3631

https://www.mdpi.com/1996-1073/15/10/3631

utolsó látogatás: 2022. november 4.

[10] 16 bites PIC mikrovezérlők energiatakarékos üzemmódjai:

https://microchipdeveloper.com/16bit:doze-idle-sleep

utolsó látogatás: 2022. november 4.

[11] Heath, Steve: *Embedded systems design*, 2003

https://archive.org/details/embeddedsystemsd0000heat

utolsó látogatás: 2022. november 4.

Ábrajegyzék

[A1] A fénykép a SZEnergy Team online galériájából származik.

[A2] https://www.ti.com/lit/an/sloa101b/sloa101b.pdf
utolsó látogatás: 2022. november 2.

[A3] https://www.mdpi.com/1996-1073/15/10/3631/htm
utolsó látogatás: 2022. november 2.

[A4] https://www.mdpi.com/1996-1073/15/10/3631/htm
utolsó látogatás: 2022. november 2.

[A5] Az ábrát SZEnergy Team szolgáltatta.

[A6] https://microchipdeveloper.com/16bit:doze-idle-sleep
utolsó látogatás: 2022. november 2.

Mellékletek

1. melléklet

```
455  void StateMachineUpdate(void){
456
457    if(!VcuState_A.DRIVE && !VcuState_A.REVERSE){
458      current_state = Neutral;
459    }else if(conversion.ADC_Word > 0){
460      if(VcuState_A.DRIVE){
461        current_state = Drive_Pedal;
462      }else if(VcuState_A.REVERSE){
463        current_state = Reverse_Pedal;
464      }
465    }else if(VcuState_A.ACC && conversion.ADC_Word == 0){
466      if(VcuState_A.DRIVE){
467        if(flags.fn2_on){
468        current_state = LUT_Ref;
469        }else{
470        current_state = Automatic_Acc;
471        }
472      }else if(VcuState_A.REVERSE){
473        current_state = Automatic_Dec;
474      }
475    }else{
476      current_state = Neutral;
477    }
478
479    if(prev_state != Neutral && current_state == Neutral){
480      mc_ref.CAN_Word = 0;
481      prev_ mc_ref.CAN_Word = 0;
482      McCommandUpdate();
483      CanMcCommand();
484    }
485    prev_state = current_state;
486  }
```

2. melléklet

```
70  if(flags.mc_command_update_and_send == true && VcuState_B.AUT == false){
71          if(!VcuState_A.MC_OW){
72              PotConversion();
73              StateMachineUpdate();
74              CalculateSpeed();
75              if(current_state != Neutral){
76                  if(current_state == Automatic_Acc || current_state == Automatic_Dec){
77                      CalculateMCRef();
78                  }else if(current_state == LUT_Ref){
79                       if(flags.first_lap_started){
80                           CalculateDistance();
81                       }
82                      GetLUTMCRef();
83                  }
84                  RateLimiter();
85                  McCommandUpdate();
86                  CanMcCommand();
87              }
88          }else{
89              PotConversion();
90              McCommandUpdate();
91              CanMcCommand();
92          }
93          flags.mc_command_update_and_send = false;
94      }
```

3. melléklet

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4
5     int buff;
6     int n;
7     FILE *fRead;
8     FILE *fWrite;
9
10 int main(int argc, char **argv)
11 {
12    if ((fRead = fopen("from.txt","r")) == NULL){
13       printf("Error! opening file");
14
15       // Program exits if the file pointer returns NULL.
16       exit(1);
17    }
18
19    if ((fWrite = fopen("to.txt","w")) == NULL){
20       printf("Error! opening file");
21
22       // Program exits if the file pointer returns NULL.
23       exit(2);
24    }
25
26    while ((buff = fgetc(fRead)) != EOF){
27       if(isdigit(buff)){
28          n++;
29       }
30       if(buff=='\n'){
31          fputc(',', fWrite);
32       }else{
33          fputc(buff, fWrite);
34       }
    }
```

```
35      fputc('\n', fWrite);
36      fprintf(fWrite,"%d",n);
37
38      fclose(fRead);
39      fclose(fWrite);
40      return 0;
41  }
```

4. melléklet

```
526 void GetLUTMCRef(void){
527     if(!flags.first_lap_started){
528         mc_ref.CAN_Word = 0;
529     }else{
530         if(flags.new_lap){
531             vehicle.distance = 0.;
532             flags.new_lap = false;
533         }
534             lut_buffer=
535 lut[((uint32_t)(vehicle.distance+DISTANCE_STEP)/LUT_DISTANCE_RESOLUTION)
536 %LUT_SIZE];
537         if(lut_buffer){
538             mc_ref.CAN_Word = 1023;
539         }else{
            mc_ref.CAN_Word = 0;
        }
    }
}
```