
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Rácz Dániel

# TESZTKÖRNYEZET FEJLESZTÉSE FORDULATSZÁMMÉRŐ SZENZOROK VIZSGÁLATÁHOZ

TANSZÉKI KONZULENS

Dr. Sujbert László

KÜLSŐ KONZULENS

Gusztáv Tamás
AUDI HUNGÁRIA MOTOR KFT.

BUDAPEST, 2013


SZAKDOLGOZAT-FELADAT

**Rácz Dániel (M74M0G)**
szigorló villamosmérnök hallgató részére

# Tesztkörnyezet fejlesztése fordulatszámmérő szenzorok vizsgálatához

Az Audi Hungária Motor Kft. elektromos laboratóriuma, a gyártás során vagy a vevőnél használat során keletkező hibás elektromos alkatrészek vizsgálatával foglalkozik. Gyakori feladat a fordulatszám-jeladók analízise és hibáinak felderítése. Ezt korábban a laborban funkcionális tesztek elvégzésével tették: oszcilloszkópos megfigyelés és autókba történő beépítés során fellépő rendellenes viselkedés vizsgálatával. A régi eszköz nem képes impulzusszélességek, szenzorpontosságok, szögelfordulás-értékek meghatározására. Ezen tulajdonságokhoz a berendezés fejlesztésére volt szükség, hogy az aktuális hibaképről minél szélesebb körű információhalmaz álljon a vizsgálatot végző személy rendelkezésére.

A hallgató feladata, hogy új mérési összeállítást tervezzen és valósítson meg mind hardveres, mind szoftveres szinten. Az eszköz legyen automatizálható, hogy hosszabb járatási tesztek is felügyelten végrehajthatóak legyenek. Referenciaként egy elegendően nagy felbontású enkóder szolgáljon, amely által szolgáltatott jelek alapján a vizsgált szenzor specifikált paraméterei (abszolút és ismétlési pontosság a különböző működési módokban, impulzusszélességek) egyértelműen meghatározhatóak. Kimenetként a program egy mérési jegyzőkönyvet állítson elő, amely használható a hibás szenzorok gyártókkal szembeni reklamációinál.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a fordulatszám-jeladó szenzorok felépítését, működésének fizikai hátterét!
- Készítsen rendszertervet a követelmények alapján, és mutassa be annak elemeit!
- Tervezzen új hardvert, amely imitálja a főtengely/vezérműtengely forgó mozgását! A hardver legyen képes szabályozható, stabil forgási profilok előállítására, biztosítsa a rendszer kiegyensúlyozottságát és nagy hőmérséklettűréssel rendelkezzen a klímakamrában történő tesztelhetőséghez.
- Tervezzen hardvert, amely alkalmas az enkóder jeleinek fogadására és feldolgozására, a vizsgált szenzorok időzítéseinek mérésére, és a mért értékek továbbítására!
- Készítsen programot, amely alkalmas a szenzorok jeleinek fogadására, a hibaanalízis elvégzésére, a mérést végző felhasználó tájékoztatására és az eredmények további felhasználásához szükséges dokumentáció elkészítésére!

**Tanszéki konzulens:** Dr. Sujbert László, docens
**Külső konzulens:** Gusztáv Tamás (Audi Hungária Motor Kft.)

Budapest, 2013. október 1.

.....
Dr. Jobbágy Ákos
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.2.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

# Tartalomjegyzék

# HALLGATÓI NYILATKOZAT

Alulírott **Rácz Dániel**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2013. 12. 17.

_____

Rácz Dániel

# Összefoglaló

A diplomamunkám a gépjárművekben található fordulatszám jeladók laboratóriumi tesztelésére hivatott mérőberendezés elkészítéséről szól. A fordulatszámmérők a főtengelyről és a vezérműtengelyről szolgáltatnak információkat a tengelyekre helyezett jeladó tárcsák segítségével. A jelek hiánya vagy téves érzékelése esetén a motorvezérlő egység nem képes a tökéletes járatás megvalósítására. A feladatom egy olyan szerkezet kitalálására és elkészítésére irányult, amellyel az Audi Hungária Kft. elektromos laboratóriumába hibásan beérkező szenzorok rendellenes viselkedései és hibás paraméterei felderíthetővé válhatnak.

A Bevezetés rövid betekintést ad az autóipar fejlődéséről, digitalizálódásáról, valamint a bonyolult szenzorhálózatokkal ellátott gépjárművek elterjedéséről és ennek szükségességéről. Ezt követi a fordulatszám jeladók motorokban betöltött szerepének, alaptulajdonságainak, működési módjainak és az ezek alapját képező fizikai ismereteknek, az egyes típusokra jellemző egyedi sajátosságoknak és a vizsgálatok tárgyát képező paraméterek fogalmi hátterének a bemutatása. Ilyen paraméterek, melyekről minden mérés során szeretnénk pontos eredményeket kapni, az érzékelt jelek szögelfordulásai, impulzusszélességei, a szenzorok abszolút és ismétlési pontosságainak értékei.

A fejlesztés a korábban alkalmazott mérőberendezés hiányosságainak megismerésével és az új rendszerrel szemben felmerülő követelmények megfogalmazásával kezdődött. Ezt követte egy rendszerterv elkészítése és a hozzá tartozó alkotóelemek legyártatása, beszerzése. Az eszköz digitális jelfeldolgozáson alapul, mely egy mikrokontrolleres egységből és a hozzá szorosan kapcsolódó LabView fejlesztőkörnyezetből áll. A Megvalósítás fejezet a munkám nagy hányadát képező programozási feladat egyes fázisait mutatja be. Végigveszi a beágyazott és grafikus elvű programkódok felépítését, az alkalmazott algoritmusokat és struktúrákat, továbbá funkcionálisan ismerteti az analízist végző személy tájékoztatására szolgáló LabView-ban írt felhasználói felületet és annak egyes elemeit. Az utolsó két fejezet az elkészült készülék továbbfejlesztési lehetőségeit és a rendszer tökéletesítéséhez szükséges hátralévő feladatokat veszi sorra, majd összefoglalja a fejlesztés egyes fázisait és a felhasználó kezébe adható rendszer elkészültéhez vezető út lépéseit.

# Abstract

My thesis is about a development of a laboratory test measurement system for tachometer sensors. The tachometers can give information about crank shaft and camshaft using encoder wheels. The engine control unit can not produce the perfect engine operation in case of lack of the signs or incorrect detection of them. My project was to find out and to create a system, with which the bad sensors' behaviors and faulty parameters can be detected.

The introduction chapter gives us short information about the evolution of the automobile industry and the complex sensor networks. This is followed by the introduction of tachometer sensors with their main properties, with the background physical knowledge and the most important parameters of sensors, which should measure the system correctly. These parameters are the angle of rotations of measured signs, the pulse width, the absolute and the repeatability accuracy.

The development began with the recognition of the previous system's deficiencies and continued with the guessing of the new system's requirements. After that came the making of a new system design, the production and the purchasing of the components. The device is based on digital signal processing, which contains an ambient microcontroller unit and a LabView development environment.

The implementation chapter shows the major part of my job, which was the programming of the two systems in two different languages: embedded C and graphical programming of LabView. The chapter introduces the structures of the codes, the used algorithms in detail; furthermore it functionally describes the user interface written in LabView.

The two last chapters present the further development opportunities and the remaining tasks for improving and finalizing the measurement system. After that it summarizes the phases of the full development process and provides a comprehensive overview about the steps which led to the completed system.

# 1 Bevezetés

## 1.1 Környezeti háttér

Az autóipar korai szakaszában még nem tekintették fontosnak a gyártók, hogy minél gazdaságosabb és környezetkímélőbb motorokat gyártsanak. Ám napjainkra ez a szemlélet megváltozott, köszönhetően a környezettudatosabb gondolkodásmódnak. Ahhoz, hogy egy gépjármű motorja megfeleljen az összes normának és előírásnak, a fogyasztás és a károsanyag-kibocsátás drasztikus csökkentésére volt szükség. Ehhez bonyolult és rendkívül összetett szenzorhálózatokkal kellett a motorokat ellátni, amelyek többek között érzékelik a szelepek és dugattyúk működési ciklusait, a hőmérsékletet, a nyomást, a fordulatszámot, a gyorsulást, a légtömeget és az oxigén-benzin keverék arányát. Ez alapján elmondható, hogy a napjainkban gyártott belső égésű motorok már nem csupán periodikus termodinamikai munkafolyamatokat végző mechanikai gépezetek, hanem összetett informatikai rendszerek is, amelyek adatgyűjtést, kiértékelést és beavatkozást végeznek a másodperc tört része alatt.

A legelsőként alkalmazott szenzorok között volt a **fordulatszám jeladó**. Ez a szenzor a motortérben található meg a fogaskoszorúnál, és a vezérműtengelyre vagy a főtengelyre helyezett jeladó tárcsáról olvassa le a tárcsa által szolgáltatott különböző impulzusok sorozatát. A fordulatszám jeladó beépítésével az elsődleges cél az volt, hogy a fogyasztás csökkenjen. A jeleket először ahhoz használták fel, hogy irányítani lehessen a gyújtáskapcsolást, majd felismerve a szenzorban rejlő lehetőségeket, később már a sűrítés, befecskendezés folyamatainak ütemezésénél is használt vették az impulzusok sorozatainak. A gyújtás vezérlése fogyasztáscsökkenéshez vezetett, és egy fontos hibajelzési lehetőséget is magával hozott, az égéskimaradás észlelését. A főtengely szögsebessége a motor normál működése során nem állandó, lassuló és gyorsuló szakaszok váltják egymást az egyes hengerek munkaütemének megfelelően. A sűrítési ütem a motor mozgási energiáját használja fel a hengerben lévő benzin-levegő keverék összepréseléséhez, ezáltal lassítja a főtengely forgását, míg a gyújtás után a munkaütem a főtengelyre gyorsító erővel hat. Ha egy hengerben a keverék nem gyullad meg, a gyorsító fázis kimarad, ennek detektálásával lehet következtetni a gyújtás hibájára. A főtengelyen lévő tárcsa jelét érzékelő szenzor információit felhasználva a motorvezérlő képes az adott ütemekhez tartozó sebességeket – és ezek különbségeit –

kiszámítani, és ezáltal a hibás működést felismerni. Manapság a fordulatszám jeladó által szolgáltatott jelekre számos alkatrész, további szenzor és vezérlőparaméter támaszkodik. Vele párhuzamosan természetesen még megannyi szenzor és újítás valósult meg (például lambda-szonda, kopogásérzékelő, EGR szelep jeladó), amelyek mind a motor minél kiszámíthatóbb és ütemezhetőbb vezérlését segítik elő.

A szenzorok – mint minden elektronikus alkatrész – meghibásodhatnak, és ilyenkor téves információkkal láthatják el a motorvezérlő logikát. Ebben az esetben a motor kihagyhat, nem biztosított az egyenletes üzem, legrosszabb esetben a motor el sem indul, vagy működés közben hirtelen leáll. Ilyenkor a hibás szenzort reklamációval visszaküldik a motorgyártó vállalathoz kivizsgálásra. A vizsgálathoz egy olyan laboratóriumi eszközre van szükség, amely képes felismerni minden olyan hibát, amelyet egy rosszul működő szenzor produkálhat. Az Audi Hungária Motor Kft-nél az volt a feladatom, hogy egy erre a célra hivatott, meglévő berendezést továbbfejlesszek a még nagyobb pontosság, még több hiba érzékelése és megkülönböztetése céljából. Az eredeti mérőeszköz egy kezdetleges, pontatlan rendszer volt, amellyel csak a legkézenfekvőbb hibák voltak felismerhetőek. Az analízis nem nyújtott elégséges információt arról, hogy a vizsgált szenzor miért nem működik helyesen, csupán a hiba megléte vagy hiánya volt érzékelhető. Ezt a mérési bizonytalanságot és pontatlanságot kellett áthidalnom és megszüntetnem új ötletekkel, valamint egy új mérési kialakítás megtervezésével és megvalósításával, hogy a fejlesztés végére egy sokkal megbízhatóbb, robusztusabb, környezeti változókra kevésbé érzékeny, és nem utolsó sorban széleskörű hibaanalízisre képes laboratóriumi vizsgálóeszközt kaphassunk eredményül.

## 1.2 A diplomamunka felépítése

A fordulatszám jeladók funkcionális ismertetése fejezetben bemutatom a régi elemző berendezés sajátosságait és hiányosságait, valamint a vizsgálat tárgyát képező szenzorok és a hozzájuk tartozó jeladó tárcsák tulajdonságait, belső felépítéseit, működési módjait és a szenzorok kimenetein megjelenő jelalakok hullámformáit. Ezzel egy átfogó képet kapunk arról, hogy a régi eszköz miért nem elég precíz mérési összeállítás a hibaanalízishez, és hogy milyen fő szempontokat kell figyelembe venni a hibák felderítéséhez, milyen paraméterek regisztrálására és számítására kell kihegyezni a méréseket. A harmadik fejezet az elkészítendő rendszerrel szemben támasztott követelményeket részletezi, mind hardveres, mind szoftveres oldalról megközelítve. Majd a tervezési folyamatba ad betekintést, hogy a felsorolt követelmények figyelembevételével milyen eszközök, kiegészítő áramkörök és alkatrészek kerültek beépítésre. A Megvalósítás fejezet pontról pontra ismerteti a rendszerterv alapján az elkészült eszközt, hogy melyik egység milyen célt szolgál, hogyan működik, valamint, hogy a megírt programkódok milyen funkciókat valósítanak meg. Az implementációt egy összegzés zárja, amely röviden összefoglalja, hogy mi volt a kitűzött cél, ebből mi az, amit sikerült megvalósítani, és hogy milyen továbbfejlesztési lehetőségeket rejt magában az elkészült összeállítás.

# 2 A fordulatszám jeladók funkcionális ismertetése

## 2.1 Fordulatszámmérés folyamata a gépjárművekben

A fordulatszámmérő szenzorok feladata az, hogy a jármű motorjában található főtengely és vezérműtengely fordulatszámát meghatározzák, egy a tengelyekre rögzített jeladó tárcsa segítségével. A tárcsáról leolvasott jelek kábelen keresztül jutnak el a motorvezérlő egységhez, amely elvégzi a megfelelő kalkulációkat és a számított értékek felhasználásával biztosítja a tökéletes motorjárást.

A belső égésű motorokban a fordulatszám és a szöghelyzet, pozíció érzékelése a főtengelyen és a vezérműtengelyen történik. A főtengely jeladók a fogas koszorú mögött található tárcsáról olvassák le az impulzusok sorozatát, amiből a főtengely szöghelyzete meghatározható. Ezek a tárcsák általában sokpólusúak, hogy az elfordulást és adott pozíciót minél nagyobb felbontásban lehessen mérni. Minden ilyen típusú tárcsán egy mechanikai indexpozíció került kialakításra, mely kijelöli a 0°-os referencia szögelfordulást (szinkronhelyet), amihez képest a többi pozíció viszonyítható. A főtengely jeladó által szolgáltatott jeleket (fordulatszám és szöghelyzet) a motorvezérlő elektronika a befecskendezéshez és a gyújtásvezérléshez szükséges további adatok kiszámításához használja.

A vezérműtengely-pozíció jeladó egy abszolút jeladó, amelyhez tartozó tárcsa a vezérműtengelyen található, annak forgásáról szolgáltat információkat. A tengely a szelepek nyitását és zárását végzi és a hozzá kapcsolódó jeladó feladata az, hogy a főtengely fordulatszám jeladójának adatait is figyelembe véve meghatározza az első henger pontos helyzetét, amivel szintén a gyújtási és befecskendezési ciklus időzítését lehet kalibrálni. A vezérműről jeleket adó tárcsán dedikált és kevés fog van, épp abból a célból, hogy az egyes hengerek mozgási ciklusai könnyen érzékelhetőek legyenek.

## 2.2 Alkalmazott szenzorok és tárcsák tulajdonságai

### 2.2.1 Szenzorok alapjellemzői

A fordulatszám jeladók közvetlenül nem érintkeznek a tárcsákkal, hanem meghatározott légrésen keresztül – katalógusérték alapján 0,7…2 mm között biztosított a helyes érzékelés – végzik a leolvasást. A szűk tűréshatárú légrés azért célszerű, mert így csökkenthetők a leolvasás közben fellépő járulékos hibák zavaró hatásai. Ilyen hatás adódhat például a motorblokk hőmérsékletingadozásából, ahol a különböző anyagok hőtágulásai befolyásolhatják a légrés nagyságát, valamint a beépítési körülményekből. Ilyen körülmény alatt értendő az a tény, hogy két azonos típusú motorban nem létezik két egyforma alkatrész és az illesztések változásai a légrés módosulásához vezethetnek.

#### 2.2.1.1 A jeladók érzékelési formái

A fordulatszámmérő szenzoroknak – működési módból adódóan – alapvetően két típusa létezik: passzív és aktív elven működő. [1]

- **Passzív szenzorok:**

A passzív szenzoroknak több fajtája is létezik (induktív elvű, Reed-relés), melyek közül a gyors működésének köszönhetően az induktív elvűt használják az autóiparban. Az induktív szenzorok állandó mágnesből és egy ezzel kapcsolatban lévő lágymágneses póluscsapból állnak (a póluscsap van szembeállítva a tárcsával), amit tekercsben helyeznek el. Így egy állandó mágneses tér jön létre. Az állandó mágneses teret a mechanikai fogakkal ellátott fém tárcsa szakítja meg a fogak és lyukak egymást követő váltása révén. A mágneses tér változása váltakozó feszültséget indukál a szenzor tekercsében (impulzussorozatot), amelynek a frekvenciája és amplitúdója arányos a kerék fordulatszámával (tehát álló tárcsánál az indukált feszültség gyakorlatilag nulla). A passzív szenzorok mára elavulttá váltak pont az amplitúdó fordulatszámfüggése miatt, hiszen impulzussorozatok feldolgozásánál az állandó amplitúdójú jelek sorozata lenne a leghasznosabb.

- **Aktív szenzorok:**

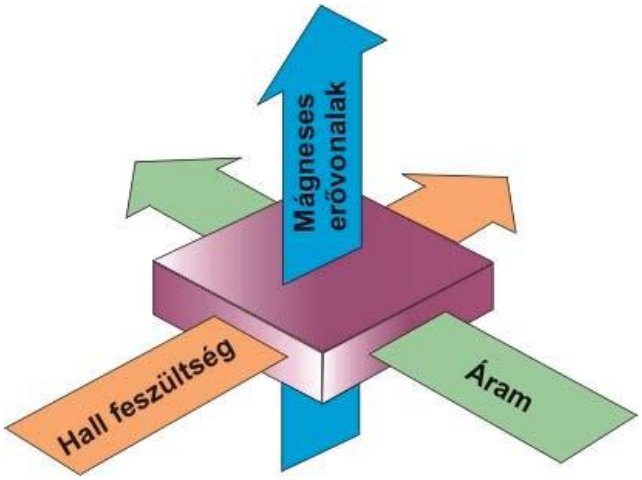
Az aktív szenzorok egy integrált logikát tartalmaznak, amely a Hall-effektus elvét használja fel a jelek érzékeléséhez. A félvezető IC a szenzor fejében helyezkedik el, és az végzi a jelek letapogatását. Az IC-k a mágneses mező legkisebb változásait is

képesek érzékelni, emiatt az induktív szenzorokénál nagyobb légrés is megengedhető. Manapság szinte kivétel nélkül minden motorban ilyen típusú szenzorok találhatóak.

Az aktív szenzoroknak létezik egy másik változata, amely optikai elven működik. Ez rendkívül egyszerű kialakítású és könnyen megvalósítható. A résekkel ellátott tárcsa egyik oldalán egy fotodióda (LED) található, amely kibocsátja a fénysugarakat, a másik oldalon egy fototranzisztor, amely fogadja a fénysugarakat. A tárcsa helyenként elzárja, majd átereszti a fényt, így biztosítva a tranzisztor vezérléséhez szükséges impulzusokat (ha fény érkezik, akkor a tranzisztor vezet, ha elállja a fény útját a tárcsa, akkor pedig lezár).
Az autóiparban nem alkalmazzák az optikai elven működő jeladókat. Az ok nagyon egyszerű: bármilyen kis szennyeződés (például olaj) is rendkívül nagy mértékben leronthatja a fotodióda fényerejét, valamint eltömítheti a jeladó tárcsa vágatait, ezáltal bizonytalanná válhat a vezérlés.

#### 2.2.1.2 Hall-effektus

Az autóiparban alkalmazott fordulatszám jeladók nagy többsége a Hall-effektus elvén működik. A Hall-effektus egy olyan jelenség, mely során a mágneses térbe helyezett áramjárta félvezető lapon az áram irányára merőlegesen a lapka egyik oldalán töltéstöbblet, a másik oldalán töltéshiány keletkezik, azaz a két oldal között feszültség mérhető (Hall-feszültség). A jelenséget a mágneses térben mozgó töltéshordozókra ható Lorentz-erő okozza. [2]




**1. ábra: Hall-effektus**

Az autóiparban széles körben alkalmazzák ezeket a fajta szenzorokat, mivel pontosabb és gyorsabb döntéshozatal lehetséges az elektronikus szenzorok beágyazott

informatikai rendszerrel történő elemzésével, mérésével, ami biztonságkritikai szempontból rendkívül előnyös tulajdonság.

Az autóiparban a következő alkalmazásokban használják a Hall-szenzorokat [3] [4]:

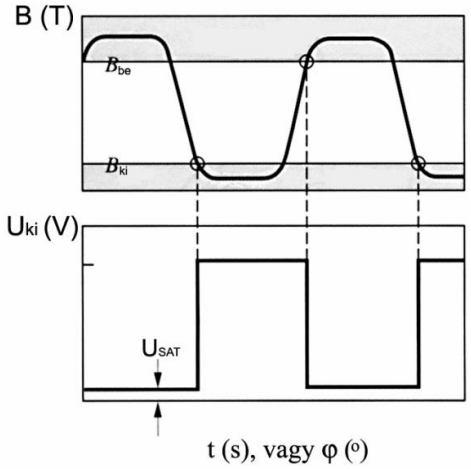
- konkrét fordulatszám mérése
- sebesség mérése
- ventilátor mozgásának érzékelése
- dugattyúhelyzet meghatározása
- blokkolás gátlásnál fékerő szabályozása
- üzemanyag szintmérése, stb.

Az Audi által alkalmazott szenzorok mindegyike egy nyomtatott áramköri lemezre ültetett Hall-IC-t[1] tartalmaz. Ez az integrált félvezető (általában CMOS[2]) chip nem egy egyszerű érzékelő, annál sokkal több feladatot lát el. Először a Hall-feszültséget erősíti, majd az IC jelformálást, stabilizálást végez (mivel félvezető alapú a szenzor, ezért hőmérsékletfüggő tulajdonságai is vannak). Működésüket tekintve a jelformáló áramkörök lineárisak, azaz a kimenetükön olyan analóg feszültség jelenik meg, amely arányos a Hall-érzékelők felületére merőleges mágneses fluxus sűrűséggel. Minden IC tartalmaz egy komparátoros logikát is, amely a bemenetére érkező analóg jeleket a kimenetén négyszögjelekké alakítja (tipikusan open-drain kimenet), biztosítva a könnyebb feldolgozhatóságot. A szenzor jelvezetéke egy felhúzó ellenálláson keresztül a táp pozitív pólusához kapcsolódik, ezáltal a végső kimenő jel a kimeneti tranzisztor állásától függően vagy tápfeszültségre van felhúzva, vagy épp földön van. Ez okozza azt a kedvező tulajdonságot, hogy a Hall-IC-k kimeneti jelfeszültségének amplitúdója független a motor fordulatszámától, ami könnyíti a feldolgozást. A digitális jelátvitelből adódóan következik az is, hogy az induktív zavarófeszültségek nincsenek hatással a kimeneti jelre, szemben a passzív (induktív) jeladókkal. Így a Hall-jeladók sokkal stabilabb és megbízhatóbb működésűek, mint az induktív jeladók, mert a szolgáltatott digitális jeleknél a komparálási szint szinte mindig ugyanoda adódik, és emiatt a komparálás nem okoz fázishibát.

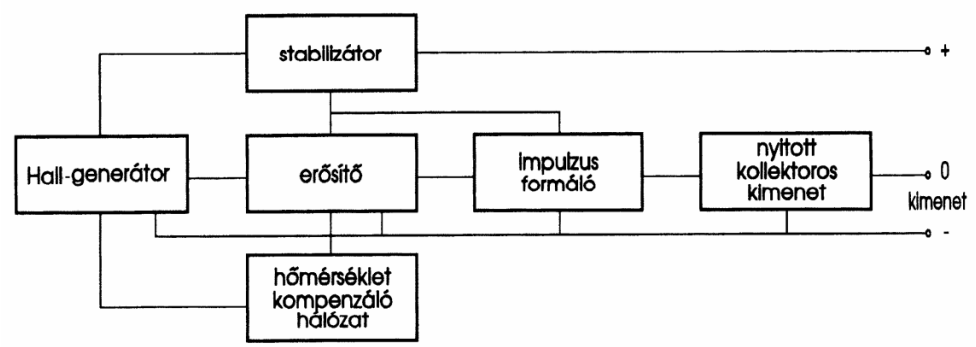
---

[1] Hall-IC: Hall-effektus elvén működő integrált áramköri elem, mely gyakran kiegészítő jelformáló logikákkal is rendelkezik (stabilizátor, erősítő, szűrő…)

[2] CMOS: Complementary Metal-Oxide Semiconductor, félvezető készítési technológia




**2. ábra: Kimeneti jelfeszültség alakulása a mágneses indukció függvényében**




**3. ábra: Egy általános Hall-IC belső felépítése és a bemenetére érkező jel útja az IC-n belül**

### 2.2.1.3 Szenzorpontosság fogalmi háttere

A szenzorok körülfordulásonként ismétlődő jelsorozatokat érzékelnek. Egy szenzor akkor működik jól, ha minden körülfordulás alatt egy bizonyos tűréshatáron belül ugyanabban a szöghelyzetben érzékeli a mágneses impulzusokat. A tűrésektől való eltérések határozzák meg a szenzorok mérési pontosságát.

A főtengely jeladók pontosságát két paraméterrel szokás jellemezni: az abszolút és az ismétlési pontosság értékével. *Abszolút pontosság* alatt a tárcsán az adott fizikai foghelyzet és a szenzor által jelzett pozíció különbségét értjük. Valójában ez még nem a tényleges abszolút pontosság, mert az egy N számú mintavételből kapott átlagérték. Ehhez a tényleges mechanikai és a szenzor által mért szöghelyzetek közti eltérések átlagát kell venni egyesével az N darab körülfordulás alatt keletkező mintákra, és az átlagtól való legnagyobb eltérés lesz az abszolút pontosság értéke.

Szemléletesen képlettel:

$$abszolút\ pontosság = \max\left[abs_{i=1\to N}\left(\frac{\sum_{i=1}^{N} S_i}{N} - S_i\right)\right]$$

,ahol $S_i$ a kijelölt impulzus szögelfordulása, amelyre az abszolút pontosságot vizsgáljuk.

Az abszolút pontosság értéke nagyságrendileg akár fokokban is mérhető, ezért a katalógusokban több fokos tűréshatárok is megengedettek (természetesen a szenzor és tárcsa típusától függően).

*Ismétlési pontosság* esetén arra vagyunk kíváncsiak, hogy N számú körülfordulás során keletkező mintavételekből mekkora a mérések átlagos tapasztalati szórása. A szenzor akkor működik hibátlanul, ha az ismétlési pontosság értéke nulla, azaz ilyenkor az átlagtól való négyzetes eltérés (azaz szórás) nulla, tehát a szenzor pontosan ugyanott érzékelte az impulzust adó fogat, mint az előző fordulat során.

A szórás képletét felhasználva az ismétlési pontosság szemléletesen képlettel:

$$ismétlési\ pontosság = 3 * \sigma = 3 * \sqrt{\frac{\sum_{j=1}^{N}\left[\left(\frac{\sum_{i=1}^{N} S_i}{N} - S_i\right)^2\right]}{N-1}}$$

(Ahol σ a szórás)

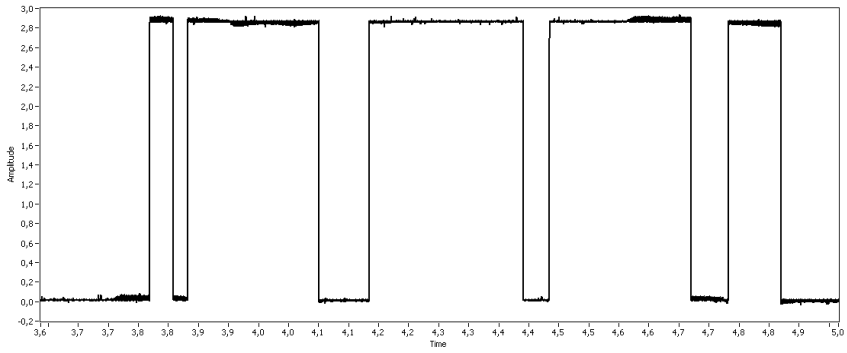
A szórás előtt található hármas szorzótényező egy konfidencia intervallumot jelöl ki, mely a normális eloszlásból adódóan a középértékétől (amely ismétlési pontosság esetén nulla) ± 3-szor szórás távolságon belül 99,7%-os valószínűséggel tartalmazza a számított értéket, és nem tekinti azt hibának.

### 2.2.2 Vezérműtengely jeladó és tárcsája

Ezen a típusú tárcsán öt mechanikai fog és öt foghézag van kialakítva. A tárcsa anyaga nem felmágnesezett fém, emiatt a leolvasást végző szenzor egy állandó mágnest tartalmazó aktív (Hall elemes) szenzor. A tárcsa fogainak (gyártásból eredő) pontossága ± 0,15', ekkora az a maximum ívhossz, amekkorával nagyobb területet fedhet le az adott fog gyártás közben a tárcsa mentén. A tárcsán nincs szinkronimpulzust jelző kialakítás, hiszen minden egyes fog mechanikai lenyomata eltérő nagyságú.

**4. ábra: Vezérműtengely jeladó szenzor és a tárcsája**

A jeladó Hall-IC-t tartalmaz, ami miatt a kimenetén létrejövő jelalak különböző szélességű, de azonos amplitúdójú impulzusok egymást követő sorozatából áll (lásd 5. ábra). A szenzor komparátoros logikával dönti el, hogy az érzékelt mechanikai fog felfutó, avagy lefutó él volt-e. A komparálási feszültség beállításához egy kezdeti szenzor-betanulási metódusnak kell lezajlania, majd amint sikeresen megtalálta a szenzor a helyes komparálási szintet, átvált normál működési tartományba. A betanulási fázis katalógus szerinti megnevezése TPO-Mód, a normál működés pedig a Dinamikus Mód, azaz a dinamikus érzékelés folyamata, a betanulást követő állapot. Minderre azért van szükség, mert minden egyes motornál, amelybe ilyen típusú vezérműtengely jeladó és tárcsa kerül beépítésre, a légrés mindig más lesz, amire a szenzorokat fel kell készíteni. A két mód segítségével lehet elérni, hogy a szenzor széles tartományokban is pontosan működjön.




**5. ábra: Egy körülfordulás alatt létrejövő impulzusok**

### 2.2.2.1 TPO-Mód

A TPO működési mód a szenzorok kezdeti állapotát jelenti, azaz a bekapcsolás utáni állapotot. Ilyenkor a Hall-IC-ben található komparátoros logika a mágneses impulzusból generált feszültségjel magas vagy alacsony szintjeiből számít komparálási szintet. Amikor elhalad a szenzor előtt egy fizikai „fog”, akkor a kimenetén logikai

nullának megfelelő feszültséget ad ki, két fizikai fog között („lyuk”, foghézag) pedig magas szintűt. A szenzor a két logikai szint megkülönböztetéséhez egy betanított kapcsolási küszöbértéket használ, melyet a gyártó határoz meg és írja bele a szenzorban található IC EEPROM-jába. A TPO-mód során új komparálási szint kerül kiszámításra, mely folyamat a hetedik elhaladó fog érzékeléséig tart. Ennyi ideje van a szenzornak beállni az üzemi működési tartományra (azaz betanulni), majd ezután módváltás következik és a szenzor a kiszámított küszöbbel Dinamikus Módban érzékel tovább.

#### 2.2.2.2 Dinamikus Mód

A nevéből adódóan dinamikus, azaz folyamatos működés közbeni állapot. Ilyenkor az átkapcsolási szint egy bizonyos értéktartományon belül ingadozik, – amely tartomány a kezdeti TPO-Módban került kiszámításra – de az impulzusonkénti számítás ugyanúgy folytatódik tovább. A dinamikus érzékelés addig tart, amíg a jelsebesség nagyobb, mint 0,35… 0,55 Hz. Ha ezen érték alá csökken, akkor ismét TPO-Módra vált a szenzor.

A két működési mód abban különbözik a leginkább, hogy TPO esetén betanulási folyamat zajlik le, lassú forgatás kíséretében, míg dinamikus módban a betanulási folyamat során meghatározott komparálási szint vezérli a fel- illetve lefutó él detektálását.

#### 2.2.2.3 Pontosság

Vezérműtengely jeladó esetén minden egyes foghoz és lyukhoz tűréshatárok lettek megállapítva a katalógusban, amelyen belül tartózkodás, illetve kívül esés jelzi a szenzor pontosságát. Jellemző hibakép e jeladóknál, hogy magas környezeti hőmérséklet esetén (100-110°C-os motorblokk hőnél) a komparálási szint megváltozik, ami az impulzusok kiszélesedését, illetve szűkülését okozza és az adott fog, illetve lyuk érzékelése kiesik a tűrésből.

### 2.2.3 Főtengely jeladó és multipolrad tárcsája

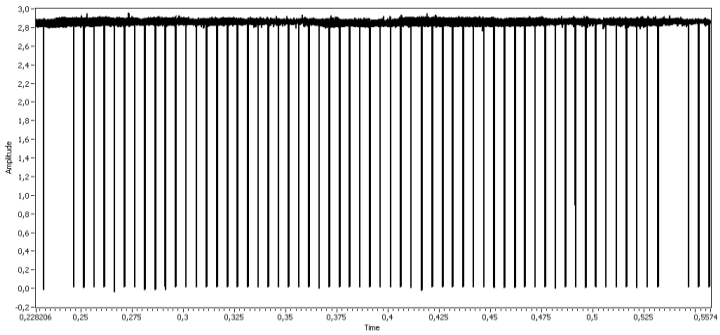
#### 2.2.3.1 Belső felépítés és szolgáltatott jelek

A főtengely multipolrad típusú tárcsája egy egészen más konstrukció a fogazott tárcsákhoz képest. Felépítése a nevéből is kiderül: multipol, azaz sokpólusú tárcsa. Alapanyaga fém, a tárcsa szélén gumírozott sávval. A gumis rész egy váltakozó északi-

déli polaritásúra felmágnesezett csík, ami összesen 58 pólusváltást tartalmaz, ebből 57 azonos távolságra lévő impulzust generál és az 58. pedig a szinkronimpulzust állítja elő. A szinkron egy dupla szélességű pólusváltás, amely a körülfordulás bekövetkeztét jelzi.

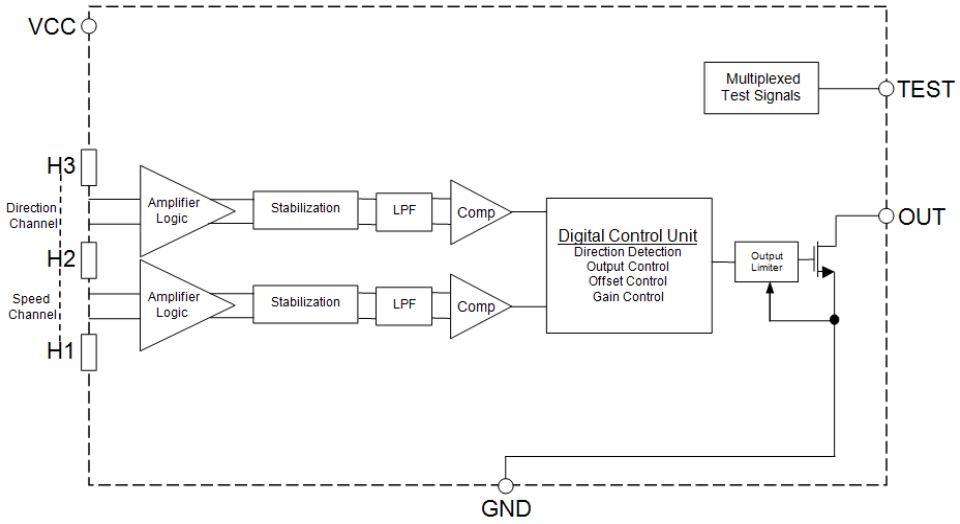
**6. ábra: Főtengely jeladó és sokpólusú tárcsája**

A tárcsához tartozó szenzor egy aktív Hall-IC-s jeladó. Ez egy fejlett, Hall-effektus elvén működő integrált áramköri elemmel ellátott fordulatszám és forgásirány detektálására képes jeladó. Az IC felépítését tekintve hasonló a 2.2.1.2 pontban részletezettel. A beérkező mágneses jelek szintén jelkondicionáló (erősítő, stabilizáló, szűrő) egységre kerülnek, majd innen a komparátoros logikára jutnak, ahol megtörténik a digitalizálás. A szenzor a csatornákra kerülő szinuszos mágneses jelek pozitív és negatív maximumaiból, a két érték átlaga alapján fogja meghatározni a komparálási szintet, ami megadja, hogy a digitalizálás során a kimeneti impulzus magas, avagy alacsony szintű lesz-e. Ezt követi a kimeneti áramkorlátozó logika, ami biztosítja, hogy ne kerülhessen az Open-Drain kimenetre meg nem engedett áramcsúcs (maximum 80 mA). A végső kimeneti jelalak impulzusszélesség modulációval előállított pulzusok sorozata. A tárcsáról leolvasott mágneses jeleknek megfelelően állandó távolságra, magas alapszintű (U_high) impulzusok követik egymást: melyekből – egy körülfordulás alatt – 57 azonos szélességű, és egy szinkronjel, amely 18° szélességű négyszögjel (lásd 7. ábra).




**7. ábra: Egy fordulat alatt keletkező 57+1 impulzus**

A 8. ábrán látható a multipolrad tárcsához tartozó főtengely jeladó szenzor belső felépítése. (A TEST ponton keresztül különböző belső analóg jeleket lehet kivezetni és megfigyelni hibakeresés céljából. Ezt a funkciót csak a gyártó használja tesztelésekhez.)




**8. ábra: Fordulatszám jeladó szenzor belső felépítése**

A mágneses érzékelésért három Hall elem felel, melyek két, differenciális mágneses jel detektálására alkalmas csatornát hoznak létre. Az egyik csatorna a Speed Channel, a másik pedig a Direction Channel. A csatornák jeleiből periódusidő számítás után sebességinformáció jut a szenzor kimenetére (OUT), a két szinuszos jel fáziskülönbségéből pedig a jeladó tárcsának és így a főtengely forgásának irányáról tájékoztat az eszköz. A tárcsa forgásirányának megfelelően a szenzor más impulzusszélességű jeleket szolgáltat. CCW[3] forgatás esetén minden impulzus 43 µsec ideig, CW[4] forgatás esetén pedig minden impulzus 88 µsec időtartamig alacsony szintű.

A forgásirány-detektálásra képes szenzorokat kimondottan a Start-stop rendszerekkel ellátott autókban használják. Amennyiben az autó megáll, az ECU[5] leállítja a motort, viszont a szenzor táplálása ekkor sem szűnik meg. Ez azért szükséges, mert a főtengely pozícióját a motorvezérlőnek folyamatosan figyelnie kell, hiszen leállítást követően a hengerekben lévő nyomás akkora nagyságú lehet, amekkora elegendő ahhoz, hogy a dugattyút az addigi mozgásával ellentétes irányban elmozdulásra késztesse, és ennek következményeként a tengely a másik irányban

[3] CCW: Counter Clockwise, óramutató járásával ellentétes irányú forgás.

[4] CW: Clockwise, óramutató járásával megegyező irányú forgás.

[5] ECU: Engine Control Unit, központi motorvezérlő egység.

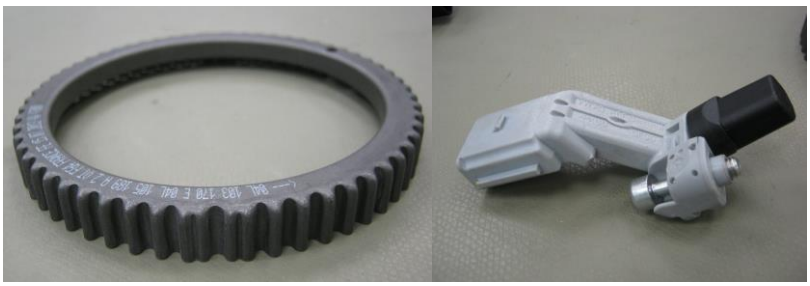
elfordulhat. Ahhoz, hogy a motorvezérlőnek ne kelljen az index jelre várnia, hanem rászinkronizálás nélkül gyors indítást hajthasson végre, ismernie kell a pontos tengelypozíciókat. A forgásirány detektáló szenzorok segítségével az ECU képes követni a leállítás alatti és utáni eseményeket is, és induláskor pontosan tudja, hogy melyik fázisból kell folytatnia a vezérlést.

#### 2.2.3.2 Pontosság

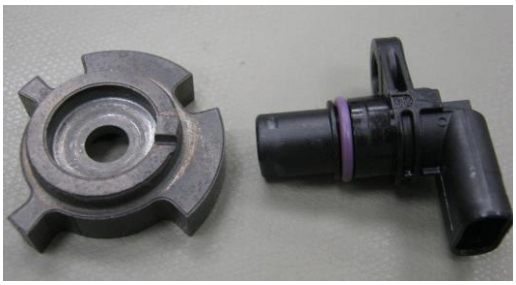
A forgásirány-detektálásra képes szenzoroknál a hibára az abszolút és ismétlési pontosságok értékeiből, és az érzékelt impulzusok szélességeiből lehet következtetni.

## 2.2.4 További szenzorok és tárcsák

Az eddig bemutatott két szenzoron és tárcsán kívül még léteznek más típusúak is, melyek nagyrészt kialakításukban térnek el, de működési módjukban és a szolgáltatott jelekben nagyban hasonlítanak a 2.2.2 és 2.2.3 szakaszokban ismertetettekre. A 9. és 11. ábrákból is látható, hogy multipolrad-jellegű érzékelésnél a gumírozott csíkon kívül létezik olyan megoldás is, amely acélból készült impulzuskereket használ. Az impulzusokat biztosító mechanikai kialakítások lehetnek vágatok (11. ábra), vagy bordák (9. ábra). Ebben az esetben az impulzusok generálásához a Hall-IC-n helyeznek el egy mágnest, amely állandó mágneses teret létesít az IC környezetében. Az impulzuskerék forgatásakor a jelenlévő, állandó mágneses tér a fogak és fogközök folyamatos ismétlődése miatt periodikusan változik, így hozva létre a négyszögimpulzusok sorozatát.



**9. ábra: Multipolrad tárcsa mechanikus fogkialakítással**



**10. ábra: Másik típusú vezérműtengely jeladó**

**11. ábra: Hall-elvű főtengely jeladó és tárcsája**

## 2.3 A fejlesztés előtt használt mérőeszköz tulajdonságai

A laboratóriumban a hibás szenzorok tesztelésére alkalmazott mérőberendezés sok olyan hiányossággal rendelkezik, amelyek felvetették egy új eszköz megtervezésének és elkészítésének az igényét. Ez az egység jelanalízis szempontjából egy rendkívül leegyszerűsített elemző készülék. Nem használ semmilyen digitális eszközt a szenzor jeleinek feldolgozásához, csupán egy motorral hajtott tengely, amelyre jeladó tárcsákat lehet felhelyezni. A különböző szenzorok pozícionálása, ezáltal a légrés precíz változtatása nehézkes, mert nem áll rendelkezésre olyan illesztő mechanika, amely kiküszöbölné ezt a problémát. A 12. ábrán látható a berendezés, amelyet a fejlesztésem megkezdése előtt a fordulatszám jeladó szenzorok analízisére használtak.

**12. ábra: A fejlesztés előtti struktúra**

A szenzorok jeleit csak oszcilloszkóp segítségével lehet mérni és megjeleníteni, amely nehézkes és időigényes, továbbá a kijelzett jelalakokból sem lehet messzemenő következetéseket levonni. Csupán annak a tényét lehet megállapítani, hogy az adott szenzor kimenetén az impulzusok formailag megfelelnek-e a katalógusban megadott hullámformáknak, vagy sem. További probléma lehet, hogy az oszcilloszkóp real-time jeleníti meg az érzékelt jeleket, és egyszerre csak egy nagyon kis tartományt képes lefedni. Emiatt előfordulhat, hogy a jelekből pont az a rész nem kerül kijelzésre, amely a hibajelenséget magában hordozza.

## 2.4 Összefoglalás

Ebben a fejezetben részletes képet adtam a szenzorok belső felépítéséről, az érzékelés fizikai hátterét biztosító Hall-effektusról, a lehetséges működési módokról és az adott szenzort, tárcsát leginkább jellemző paraméterekről, pontossági adatokról. Továbbá röviden ismertettem az általam tervezett és megvalósított berendezés előtt használt eszközt, rámutattam a hiányosságaira, amelyek az új rendszer megtervezéséhez és elkészítéséhez vezettek. A következő fejezet a tervezés fázisait veszi sorra, kezdve a mérendő paraméterekhez szükséges részegységekkel szemben fennálló követelmények megfogalmazásával, majd folytatva a kritériumokból felépülő rendszerterv bemutatásával.

# 3 Tervezés

## 3.1 Követelmények

A rendszer egy teljesen új fejlesztés, a régi eszközhöz képest minden tekintetben újragondolt koncepció, amely alapos tervezést igényelt ahhoz, hogy megfeleljen a vele szemben támasztott követelményeknek. A Követelmények fejezetben részletezem, hogy milyen eszközökre van szükség, és hogy ezek milyen specifikációknak kell, hogy megfeleljenek.

### 3.1.1 Meghajtás, szervomotor

Mivel a belső égésű motor főtengelyének, illetve vezérműtengelyének forgását szeretnénk reprodukálni, ezért a rendszerben biztosan szerepelnie kell egy motornak, amely a tengely meghajtásáért felel. A forgó mozgás létrehozásához és forgás paramétereinek változtatásához egy olyan szervomotorra van szükség, ami képes különféle felfutási ciklusok, mozgási/forgási profilok megvalósítására. Ilyen profil lehet például egy motorindítás, amikor nem mindegy, hogy a kívánt fordulatszám elérése milyen gyorsan következik be, vagy a TPO-Mód tesztelése, amihez a motornak képesnek kell lennie adott pozícióból indulva egy meghatározott új pozícióba történő megérkezésre.

A hosszú tesztelések állandó és alacsony fordulaton történnek, mert a feldolgozási pontosság ekkor maximális, viszont a szenzorok teljes működési tartományának lefedéséhez célszerű olyan meghajtást választani, amely az akár 8000-10000 fordulat/perc-es fordulatszám elérésére is képes. A 2.2.3 pontban részletezett főtengely jeladó például a 0-9000 fordulat/perc-es fordulatszám-tartományban érzékel. A motornak olyan tengellyel kell rendelkeznie, amely elviseli a tárcsák és a hozzájuk tartozó felfogató alumínium „pogácsák” súlyából adódó oldalirányú nyomatékot, fordulatszám-csökkenés nélkül. A belső égésű motort „szimuláló” készüléket klímakamrában kell működtetni, hogy a szenzorok működését a valóságban előforduló hőmérsékleti viszonyok mellett lehessen a vizsgálni. Tehát a szervomotornak ki kell bírnia a 80-110 °C-os meleget és a téli hónapok szimulálásához az akár -40 °C-os hideget is.

### 3.1.2 Referenciaszenzor

A szenzorok jeleit olyan komplex mérési eljárásokhoz szeretnénk felhasználni, amelyek meghatározzák a dokumentációkban szereplő olyan fogalmakat és számértékekkel megadott paramétereket, mint például az impulzusok szélessége, a tárcsán lévő mágneses impulzusokat szolgáltató fogak szögelfordulásai, vagy az abszolút és ismétlési pontosságok. Ezek közül a pontosságok oszcilloszkóppal nem meghatározhatóak, és a többi érték is csak a megjelenítési tartományban elemezhető a kijelzőn. További problémát jelenthet, hogy a hosszú ideig tartó mérések esetén az oszcilloszkóp véges méretű mintatároló memóriája sem biztos, hogy képes eltárolni az összes beérkező impulzust. Ahhoz, hogy számíthatóak legyenek a fent említett értékek, és hogy a szenzor által érzékelt szöghelyzetekről meg tudjuk állapítani, hogy a szenzor tényleg a valóságnak megfelelő pozíciót méri és közli, szükség van egy referenciaszenzorra, más néven referencia enkóderre. Ennek az enkódernek a meghajtott tengelyre kell kapcsolódnia, hiszen így tud az elfordulásokról referencia információt szolgáltatni. Az enkóderek legfontosabb tulajdonsága a felbontás. Amennyiben a felbontás N, az azt jelenti, hogy egy körülfordulás alatt N darab impulzust generál, tehát a teljes rendszer 360°/N felbontás mellett vizsgálható. Ezért célszerű olyan enkódert választani, amelynek felbontása elegendően nagy ahhoz, hogy az általa szolgáltatott jelek alapján a szenzorok katalógusértékei egyértelműen és precízen meghatározhatóak legyenek.

### 3.1.3 Mikrokontrolleres feldolgozó egység

A szenzor kimenetén megjelenő impulzussorozat feldolgozásához beágyazott mikrokontrolleres egység alkalmazása célszerű, mert segítségével széleskörű analízis elvégzésére nyílik lehetőség. Egy olyan korszerű MCU-ra[6] van szükség, amely elegendően gyors a folyamatosan, nagy sebességgel érkező impulzusok kezelésére. Egy főtengely jeladó 8000 fordulat/perc-es fordulatszám mellett az 58 impulzussal rendelkező tárcsáról másodpercenként (8000*58)/60 ≈ 7734 pulzust érzékel, tehát egy olyan mikrokontrolleres feldolgozó egységre van szükség, amely perifériánként legalább 8 kHz-es jelet képes fogadni és feldolgozni.

[6] MCU: Microcontroller Unit

A szenzor jeleit valamilyen referenciaértékhez kell viszonyítani, hogy az eltérésekről megfelelő információkat kaphassunk. A pontos referenciákat a kvadratúra enkóderes egység biztosítja, melynek a jeleit szintén a mikrokontrolleres áramkörre kell vezetni, hogy a processzor elvégezhesse az összehasonlításokat a szenzorok jelei és a referenciaértékek között. Ha egy N=2000 felbontású enkódert 8000 fordulat/perc-es fordulatszámon használunk, akkor percenként 2000*8000 = 16 millió impulzus keletkezik és jut az MCU perifériájára. Ez hozzávetőlegesen 267 kHz-es jelnek felel meg. Ebből következik, hogy a processzornak kvadratúra enkóderes jelek fogadásához szükséges áramkörre van szüksége, továbbá, hogy minél pontosabban szeretnénk hibát detektálni, annál nagyobb felbontású enkóderre van szükség, ami nagyobb méretű memória és regiszter egységeket, és gyorsabb kontrollert igényel.

Mivel különböző kitöltési tényezőjű négyszögimpulzusok sorozatának fogadását kell megvalósítani, ezért az összes bemenetre kerülő jelet célszerű timer-ekkel[7] fogadni és feldolgozni. Az impulzusok „elkapásához" elegendő számú időzítő/számláló egység kell, valamint egy olyan kiterjedt megszakításkezelő, amely képes többszintű rutinokat összeakadás menetesen lekezelni.

A fő feladatokon kívül (jelek fogadása, számítási műveletek végzése) a kontroller informálja a felhasználót a változók aktuális értékeiről és a mérés menetéről, valamint eredményéről. A tájékoztatást legegyszerűbben kijelző segítségével lehet megvalósítani, tehát az MCU-nak rendelkeznie kell olyan egységgel, amely kijelző vezérlésére is képes. Ezen felül legyen alkalmas soros port kommunikáció lebonyolítására is az adatok további, még gyorsabb feldolgozhatóságához (PC-re történő továbbításhoz).

A felsorolt követelményeken kívül fontos szempont az is, hogy a kontroller elérhető, könnyen beszerezhető legyen, egyszerűen kezelhető fejlesztőkörnyezettel.

[7] Timer: a mikrokontrolleren belül található olyan időzítő egység, ami felépítését tekintve egy fel/le számláló, amely órajel-frekvenciával azonos, vagy leosztott sebességű számlálásra képes. Használható események bekövetkezési gyakoriságának nyomon követésére, időzítési műveletek megvalósítására. Beágyazott rendszerekben a megszakítási rutinok nélkülözhetetlen eszközei.

### 3.1.4 Jelillesztő elektronika

A szenzorok jeleinek feldolgozhatóságához szükséges a szenzorjel után egy illesztő áramkört építeni. Az áramkörnek két fő funkciót kell megvalósítania a szenzorok adatlapjaiban specifikált paramétereknek megfelelően. Egyszerre funkcionáljon ellenállásokból felépített feszültségosztóként, hogy kezdetben egy ellenállással az 5 V-ra felhúzott amplitúdójú jeleket a mikrokontrollerek által is fogadható 3,3 V-os jelekké alakítsa, valamint kondenzátoros aluláteresztő szűrőként, hogy a parazita nagyfrekvenciás zavarjeleket kiszűrje. Így egyszerűen megoldható a digitális jelek digitális jelfeldolgozó egységre vezetése.

### 3.1.5 Szoftver

A rendszer komplexitását tekintve a szoftveres egységet két különálló blokk-ként kell kezelni. Az egyiknek a szenzorból érkező impulzusok feldolgozásával és a számított adatok továbbításával, a másiknak pedig a felhasználó tájékoztatására hivatott interfész működtetésével, és a különböző vezérlések és beállítási lehetőségek lekezelésével kell foglalkoznia.

A mikrokontrolleres egységre írt programkódnak három fő feladatot kell ellátnia: a szenzor jeleinek fogadását, ezek alapján különböző számítások elvégzését és a számított változók továbbítását a felhasználói interfészre. Mivel az impulzusokról nyerhető információk szolgálnak alapul a hibaanalízisnél, ezért olyan részmodulokra, függvényekre van szükség, amelyek a fel- és lefutó éleket egymástól függetlenül képesek kezelni. A különféle enkóder- és szenzorimpulzusok feldolgozásához megszakításkezelő szubrutinokat kell alkalmazni, mert ezzel a megoldással egymástól időben elszeparálhatóak az események, a folyamatos impulzusáramlás „lelassítható” és részekre bontható. Így megkönnyíthető a feldolgozás és a számítási műveletek ütemezése.

A program megírásakor szempont, hogy a 2.2 pontban bemutatott öt szenzorra működjön. Ehhez célszerű olyan megvalósítást alkalmazni, amely menürendszeren alapszik, és a felhasználó választásának megfelelő szegmens futtatásával és a többi szegmens inaktiválásával biztosítja a modulokra osztott, csak az adott feladat végrehajtását megcélzó beágyazott programozói szemléletmód követését. Ennek a betartásával nincs lefoglalva fölösleges erőforrás, idő és memória takarítható meg.

A sok adat megjelenítéséhez és regisztrálásához egy másik szoftver is szükséges (célszerűen PC-s környezetben), amely felhasználói felületet valósít meg. Olyan grafikus felület kialakítása a cél, mely alapján a berendezést használó tájékozódhat a mérés állapotairól, a különböző változók alakulásairól, esetlegesen beavatkozhat és paramétereket módosíthat. Nyíljon lehetőség a mérések során keletkező adatok regisztrálására, elmentésére, és szabványos információs lap (riport felület) elkészítésére, amely igazolja a vizsgálati eredményeket és referenciaként szolgál az esetleges reklamációkkal szemben.

### 3.1.6 Mechanika

A jelek feldolgozásának a pontossága mellett elengedhetetlen fontosságú a mechanikai egység pontossága is, ugyanis a legtöbb mérési pontatlanság az illesztésekből, a rezgésekből, kiegyensúlyozatlanságból eredő kotyogásból, valamint a szenzor és a tárcsa közti légrés nem megfelelő beállításából adódhat. A szenzor mindig egy adott légréshez tanul be, ezért mindenképp egy olyan mechanikai egység megtervezésére és megépítésére van szükség, amely a precíz illesztéseknek köszönhetően képes biztosítani, hogy analízis közben a motor működése és a tengely forgása által gerjesztett rezgések minél kisebb mértékben tudják befolyásolni a légrést. Ennek elkerüléséhez olyan mechanikai pozícionálás és rögzítés szükséges, amely biztosítja, hogy a légrés precízen beállítható legyen, hogy az ebből fakadó mérési bizonytalanságot kiküszöbölhessük.

Minden szenzornak saját tárcsája van, amelyek felhelyezését és leszerelését minél egyszerűbben kell megvalósítani, de ügyelve arra, hogy stabil rögzítés tartsa őket a helyükön, hogy még nagy fordulatszámon is biztonságosan lehessen használni a rendszert. A szerelés viszont olyan módon történjen, hogy minél kisebb mértékben kopjanak az egymáshoz kapcsolódó elemek, mert a súrlódás, amortizáció által keletkező hézagok lötyögést idézhetnek elő, ami csökkenti a rendszer pontosságát és biztonságosságát is.

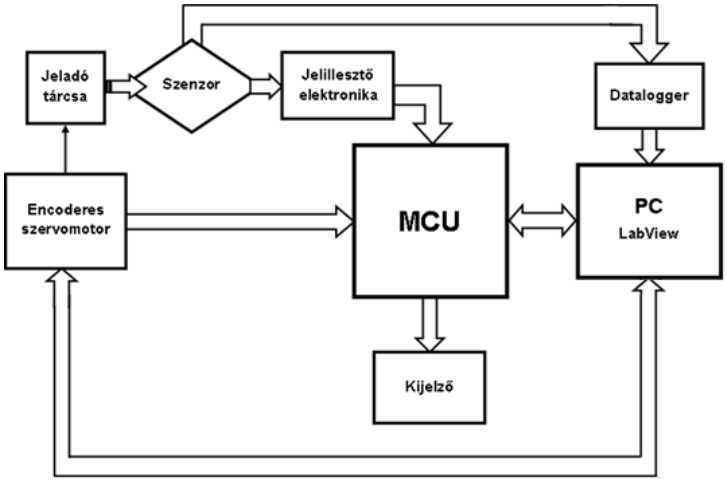
A forgatáshoz olyan erőátviteli rendszert kell tervezni, amely a nyomatékot egy az egyben adja le a motor tengelyéről, biztosítva a tökéletes és késleltetésmentes indítási, járatási és leállítási fázisokat.

Az egész mechanikai rendszert olyan anyagokból kell legyártatni, amely bírja a szélsőséges hőmérsékleti viszonyokat és hirtelen hősokkokat (gyors váltás a -40 °C és

+110 °C között). Az összes alkatrésznek a hőtágulási együtthatója közel azonos legyen, így nem fordulhat elő olyan probléma, hogy a hőmérsékletváltozás hatására az illesztések elmozdulnak. Mivel különböző hőmérsékleteknek lesz kitéve a rendszer, a gyors váltások hatására pára csapódhat le a berendezésen, ezért fontos odafigyelni arra is, hogy az alapanyagok a korrózióval szemben ellenállóak legyenek.

## 3.2 Rendszerterv

A követelmények megfogalmazódásával párhuzamosan készült el a rendszerterv is, amely összefogó képet alkot a megvalósítandó – és végül megvalósított – berendezésről. Az alábbi ábrán a legfontosabb alkotóelemek megjelölésével felvázolt blokkséma látható. Minden alkatrész közvetett vagy közvetlen kapcsolatban áll a rendszer magját képező mikrokontrolleres egységgel. Ez a processzor a lelke az egész szerkezetnek, minden adat keresztül megy rajta, minden érzékelt impulzust beolvas, feldolgoz, kiértékel. A rendszerterv egyetlen mechanikus egysége a kvadratúra enkóderrel ellátott szervomotor és a tengelyére kapcsolódó, megforgatandó jeladó tárcsa. Az összes többi elem elektronikus és digitális, mivel már a szenzor is digitális kimenetet szolgáltat. Ez alól kivételt képez az illesztő logika, amely beállítja a vezérlőegység bemeneteire kerülő jelek feszültségszintjeit, hogy a túlfeszültség ne tehessen kárt a mikrokontrollerben.




**13. ábra: Rendszerterv**

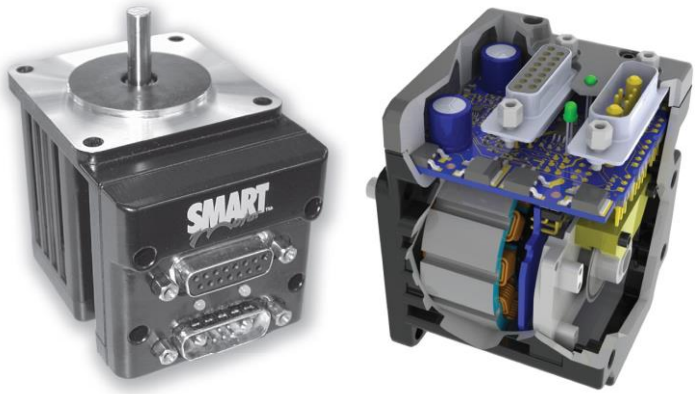
A 3.2 fejezet alpontjai a rendszerterv azon elemeit veszik sorra, amelyek több gyártó termékei közül kerültek kiválasztásra, és magyarázatot adnak arra, hogy mik azok a tulajdonságok, amelyek alapján a tervezés során az adott eszköz mellett döntöttem.

### 3.2.1 Szervomotor

A készülő rendszerbe legjobban illő motor keresése közben több szempontot is szem előtt kellett tartani. A legelső fontos paraméter a fordulatszám volt. A középkategóriás (és így az olcsóbb) ipari szervomotorok közül szinte egyik sem képes 8-10000 fordulat/perc-es fordulatszámot produkálni, vagy amelyik képes, az olyan kis méretű és ezáltal olyan kicsi tengelyterheléssel rendelkezik, hogy nem tudja a jeladó tárcsák meghajtását megvalósítani. A középkategória feletti szervomotorok – mint például a Siemens SIMOTICS S szériájába tartozó 1FK7 motor [6] – sokkal erősebb és nagyobb nyomaték leadására képes tengellyel rendelkeznek (1-10 Nm), robusztusak, nagy méretűek, viszont a kitűzött 8-10000 fordulatot ezek sem képesek elérni. Az ilyen típusú ipari szervókat a gyártósorokon szokták alkalmazni, ahol nem olyan fontos a nagy fordulatszám, de a strapabíróság, megbízhatóság, nagy hőmérséklettűrés és a hosszú élettartam elengedhetetlen. Nagy hátrányuk ezeknek a motoroknak, hogy kiegészítő meghajtó egységre van szükség a működtetésükhöz. Ezek a szabályozók összetett rendszerek, melyek magukban foglalják a PLC-s[8] vezérlést, a különböző feszültségtartományokat előállító tápegységet és a biztonsági funkciókért felelős áramköröket is. A szervomotor és a hozzá tartozó vezérlő együtt már nagyon költséges (több ezer euró), és van egy további probléma is: a táplálás. A 230…480 VAC váltakozó áramú tápfeszültség az ipari környezetben megszokott, ám laboratóriumban – ahol főleg törpefeszültséggel működő szenzorokat vizsgálnak – egy analízis berendezésnél nem célszerű. Sokkal használhatóbbak az alacsony egyenfeszültséggel táplált motorok, mert nincs szükség speciális átalakító és illesztő egységre, egy közönséges labortáppal megoldható az áramellátás.

Az előbbi rövid gondolatmenet és kutatás segített rátalálni a 14. ábrán látható Moog Animatics által gyártott SmartMotor SM23165D típusnévre hallgató szervomotorra. A motor saját tömege alig haladja meg a fél kilogrammot, nagysága pedig csupán 5,8*5,8*7,9 cm. Névleges, terheletlen fordulatszáma 10400 fordulat/perc, ami jóval az átlag feletti. A motor fordulatszámtól függően 12…48 VDC táplálást igényel, ami könnyen biztosítható.

[8] PLC: Programmable Logic Controller, programozható logikai egység. Ipari szabályozástechnikában, villamos, vagy villamosan működtetett folyamatokban alkalmazott folyamatirányító berendezések vezérlésére, mérésadatgyűjtésre és még sok más ipari feladatra használják. [7] [8]




**14. ábra: SmartMotor SM23165D [5]**

A motor kiválasztásában döntő szerepet játszott, hogy belső, programozható mikrokontrolleres vezérlőegységgel rendelkezik. A benne található digitális egység egyben szabályozza a táplálást és saját PID[9] szabályozója segítségével vezérli a tengely forgatását. A motor ezen kedvező tulajdonsága nagy mértékben befolyásolta a fejlesztés időtartamát, hiszen nem volt szükség külső panelra, drága szabályozóra, mely működésének és felprogramozásának a megtanulása hosszú időt vett volna igénybe. Az SM23165D kontrollere RS232/RS485 soros porton keresztül képes kommunikáció lebonyolítására a PC-n futó vezérlő szoftverrel és így online programozhatóvá teszi a motort. A megoldással akár forgás közben is változtathatunk a szögsebesség, gyorsulás értékein és még sok egyéb paraméteren. A gyártó ingyenes, kimondottan a motorra fejlesztett saját fejlesztőkörnyezetet is biztosít, amellyel a programozás, a működtetés és a PID szabályozó paraméterezése egyszerűen megoldható.

A SmartMotor tehát egy összetett mechanikai és digitális rendszer, amely magába foglalja mind a meghajtást, mind a vezérlést és a kommunikációt megvalósító egységeket. Kialakítása az ipari szabványokat követi, hőmérséklettűrése -40-től +110°C-ig terjed, tehát a klímakamrás tesztekhez kiválóan alkalmazható.

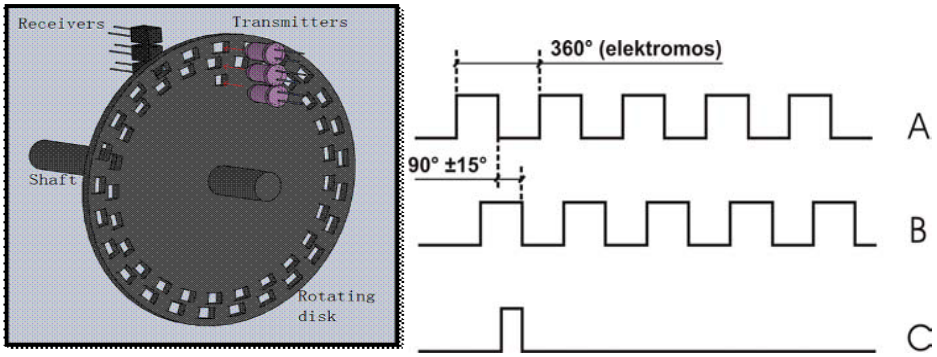
[9] PID szabályozó: egy lineáris rendszerek szabályozásánál gyakran alkalmazott, párhuzamos kompenzáción alapuló szabályozótípus. A PID rövidítés a szabályozó elvére utal, a szabályozó által kiadott végrehajtójel

- a hibajellel (P: *proportional*),
- a hibajel integráljával (I: *integral*), valamint
- a hibajel változási sebességével, deriváltjával (D: *derivative*) arányos tagokból adódik össze, azaz a végrehajtójel a jelenlegi hiba, a múltbeli hibák és a várható hibák függvénye. [9]

## 3.2.2 Kvadratúra enkóder

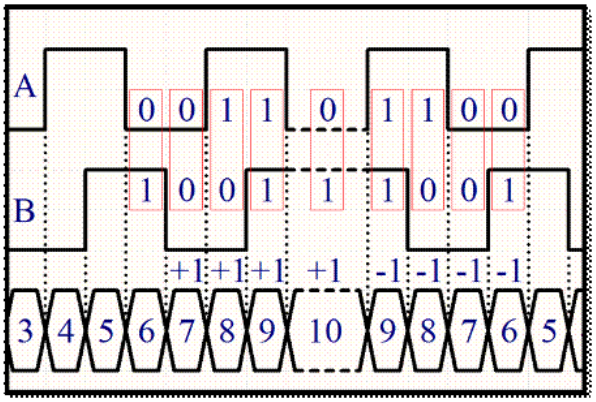
### 3.2.2.1 Működése és alapvető jellemzői [10] [11]

Az enkóderek olyan szenzorok, amelyek egy tengely elfordulását érzékelve az elfordulás szögével arányosan digitális jeleket szolgáltatnak. Több fajtája is létezik, a SmartMotor-ban található egy optikai érzékelésen alapuló inkrementális kvadratúra enkóder. A jeladóban egy tengely forog (amely a motor tengelyéhez kapcsolódik), amihez belül egy tárcsa van rögzítve. A tárcsa anyaga általában átlátszó üveg, vagy perforált fém, amire a szélén apró rovátkákat visznek fel. Egyik oldalán egy fotodióda (LED fényforrás), a másik oldalán pedig egy fényérzékeny fototranzisztor található. Miközben forog a tárcsa, a lyukak a fényt hol kitakarják, hol áteresztik, így az érzékelőre fényimpulzusok sorozata érkezik, amelyből a vevő egységben ennek megfelelő elektromos impulzusok keletkeznek. Az enkóderek legfontosabb tulajdonságát, a korábban már említett felbontást a tárcsa határozza meg, vagyis az, hogy a tárcsán hány darab impulzust generáló rovátka található. Minél több, annál több impulzus érkezik egy körülfordulás alatt (és természetesen annál nehezebb az előállítása is), és annál kisebb szögelfordulás az, amit az enkóder képes érzékelni. A SmartMotor-ban helyet kapó enkóder inkrementális elven működik. Az inkrementális (növekményes) forgó jeladókban olyan tárcsa található, amelyen egymástól azonos távolságra egyforma méretű vájatokat alakítottak ki. A lyukakon átszűrődő fényt két darab fototranzisztor figyeli. Ezek úgy vannak elhelyezve, hogy a tárcsa forgásakor egymáshoz képest 90°-kal eltolt fázisú jeleket adjanak. A két jel az „A” és a „B” csatornát határozza meg. Mindkét csatornán 50%-os kitöltési tényezőjű négyszögjelek sorozata jelenik meg, köszönhetően az enkóderben található jelformáló logikának. Ahogy a 15. ábrán is látható, az „A” és „B” csatornákon kívül létezik még egy harmadik, kitüntetett szerepű csatorna is, a „C” csatorna. Ez a tárcsa egyetlen rovátkájából keletkező impulzus a zérus, vagy indeximpulzus. Segítségével a teljes körülfordulás bekövetkezte detektálható.




**15. ábra: Inkrementális enkóder tárcsája és a három csatorna által kiadott jelek**

Ha az inkrementális enkóder jeleit szeretnénk hasznosítani, akkor egy mikrokontrolleres logikára van szükség. Illesztést követően a feldolgozó egységben egy kijelölt számláló tartalma növekszik, illetve csökken attól függően, hogy milyen az „A” és „B” csatorna egymáshoz viszonyított fázishelyzete, azaz a forgás iránya, amiből következik, hogy a számláló tartalma arányos a tengely elfordulásával. Attól függően, hogy mely élváltásokat figyeli a mikrokontroller, az enkóder felbontása akár meg is négyszerezhető (quadratic count, innen ered a kvadratúra kifejezés). Ez úgy érhető el, hogy a számláló egység minden egyes szintátmenetnél növeli vagy csökkenti tartalmát (lásd 16. ábra).




**16. ábra: Négyszeres felbontás**

Kvadratúra enkódert elsősorban elmozdulás mérésére, pozíció érzékelésére, pozícionálásra használhatunk.

### 3.2.2.2 Felhasználása, szerepe a rendszerben

A referencia enkódert leggyakrabban a motor által forgatott tengelyre helyezik el, így biztosítva a tengely forgásakor keletkező pozícióváltozások figyelését. A SmartMotor SM23165D azért egy célszerű választás a 3.2.1 pontban felsorolt tulajdonságok mellett, mert tartalmazza a referenciaszenzorként szolgáló kvadratúra

enkódert is. Felbontása 4000 inkrement/fordulat, amely 360°/4000 = 0,09° pontosságot eredményez. Tehát kevesebb, mint egy tized fokos szögelfordulást képes érzékelni. Az „A” és „B” csatorna jelei közvetlenül a motor kimeneti portjaira vannak vezetve, amelyeket egy kvadratúra enkóderes jelek fogadására specializálódott mikrokontroller timer-ének megfelelő bemeneti GPIO portjaira köthetünk. Egy ilyen timer képes különbséget tenni az élváltások között (fel- vagy lefutó), valamint azonnal fáziskülönbséget is számít a két jel fázisaiból. Az időzítő/számláló egység inicializálását követően a megírt beágyazott program egyik irányba történő forgatáskor növelheti a számláló értékét, másik irányban pedig csökkentheti.

A referencia enkóder az egész mérési folyamat legfontosabb eleme, a legtöbb érték számításánál az enkóder impulzusait figyelő számlálók tartalmát használjuk fel:

- körülfordulás bekövetkeztének jelzéséhez,
- körülforduláskor az adott változók alaphelyzetbe állításához,
- a PC-re átküldendő adatok strukturálásához,
- a soros porton történő kommunikációhoz szükséges szinkronkarakter kiküldésének időzítéséhez, és ezzel egyetemben az adatcsomagok továbbításának időzítéséhez,
- a szenzorok által érzékelt impulzusok helyességének megállapításához: tehát ahhoz, hogy az egy élváltás elkapásakor regisztrált inkrementszám és a tényleges mechanikai fog helyzetéből adódó inkrementszám között mekkora eltérés tapasztalható,
- a tengely fordulatszámának számításához,
- az egyes impulzusokhoz tartozó szögelfordulások meghatározásához,
- az abszolút és ismétlési pontosságok számításához.

## 3.2.3 MCU és fejlesztőpanel

### 3.2.3.1 STM32F103 MCU

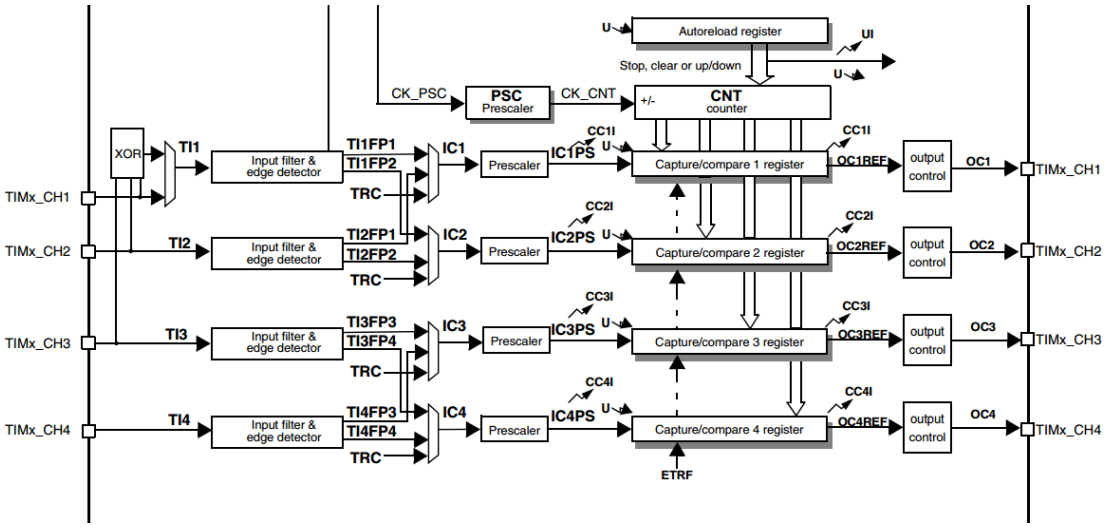
A korai tervezési fázisban két processzorcsalád terméke volt számomra rendkívül meggyőző, közülük választottam ki a végleges kontrollert. Az egyik az ST Microelectronics által gyártott STM32F103, a másik pedig a Texas Instruments Stellaris LM4F232H5QD processzora volt. A két jelfeldolgozó hasonló paraméterekkel bír, mindkettőt kimondottan kvadratúra enkóderes jelek fogadásához és automatizálási

műveletekhez ajánlja a gyártó. A két processzor jelfeldolgozási sebessége közti különbség a feladat megoldását tekintve nem befolyásoló tényező, ugyanis a Texas terméke
80 MHz-es Cortex-M4 maggal van ellátva, amely a 72 MHz-es Cortex-M3 magú STM32F103-hoz képest nem jelent sokkal gyorsabb munkavégzést. Az újabb generációs mag (M3 helyett M4) csupán bonyolultabb architektúrát eredményez, amely nem szükséges, hiszen a funkciókat, amelyeket a feladat során használni kell, mind a két kontroller képes megvalósítani. Emiatt, és a 3.1.3 fejezetben részletezett feltételek alapján esett a választás az ST Microelectronics STM32F103 mikrokontrollerére. Ez egy ARM architektúrájú Cortex-M3 magra épülő 32 bites vezérlő egység. Az órajel elegendő ahhoz, hogy még a lassú, sok ciklusú aritmetikai műveleteket is (mint például a szorzás, osztás) gyorsan végrehajthassa a kontroller, köszönhetően a beépített hardveres osztó áramkörnek és az egy órajelciklus alatt végbemenő szorzásnak.

Az STM32F103 minden olyan funkcióval, hardveres egységgel és beállítási móddal el van látva, amely egy mai, korszerű mikrokontrollertől megkövetelhető. Található benne DMA-vezérlő[10], AD/DA átalakítók, megszakításkezelő egység, összesen 14 darab 16 bites timer, I2C, SPI, USART, CAN, USB és Ethernet kommunikációs modulok, kijelző vezérlésére alkalmas modul. A szenzorok jeleinek feldolgozása szempontjából a szűk keresztmetszetet a számláló egységek jelentik. A kontrollerben három fajta timer foglal helyet: kifejezetten speciális szabályozásokhoz alkalmazható (Advanced-control), általános célú (General-purpose) és alapvető működési módok megvalósításához használható (Basic) timerek. A speciális célú időzítők annyival tudnak többet, hogy hatlépcsős PWM jelek generálására és különféle break-műveletek megvalósítására is képesek. A speciális és az általános célú is képes enkóder jeleinek fogadására, beállítástól függően adott jelátmenetű impulzusok fogadására és összehasonlítására, és még sok más egyéb funkcióra. A Basic timer egy egyszerű számlálót valósít meg, amely a megadott frekvenciaosztási aránynak (Prescaler) megfelelően adott értékig (Period) elszámol, majd túlcsordul és kezdi elölről a számlálást.

[10] DMA: Direct Memory Access, közvetlen memória hozzáférést biztosító vezérlőegység, amely lehetővé teszi, hogy az adatok a perifériák és a memória között közel olyan sebességgel áramolhassanak, mint memória és memória között.

A 17. ábrán látható egy általános célú időzítő blokkvázlatának egy fontos részlete, amely a bemenetre érkező jelek fogadását és további műveletekre való előkészítését valósítja meg.




**17. ábra: STM32F103 egy timer-ének blokkvázlat részlete**

Minden timer 16 bites, fel- és lefele számlálásra képes. A 16 bites órajelosztó (Prescaler) segítségével a számlálók órajele[11] 1 és 65 535 közötti számmal leosztható, így a kívánt frekvencián alkalmazható az előírt feladatra. Összesen négy, egymástól független csatornával rendelkeznek (ahogy azt a 17. ábra is illusztrálja), melyek négy különböző módban működhetnek:

- *Input Capture mód:* a bemenetre érkező impulzusok közül beállítástól függően a fel- vagy lefutó él érzékelésekor egy jelzőbit (Flag, 17. ábrán CCxI jel) bebillen. Az élváltásokat a bemeneti éldetektor különbözteti meg egymástól, majd ha az adott csatornához tartozó élváltás jön, akkor azt a Capture/Compare regiszterbe továbbítja, mint eseményt. Erre az eseményre megszakítás generálható, amely további műveletek elvégzésére ad módot. A jelzőbit értéke szoftveresen törölhető, majd újbóli elkapáskor ismételten bebillen.
- *Output Compare mód:* ez a funkció kimeneti hullámforma változtatására, vagy a számláló egy adott értékénél valamilyen jelzésre használható. A Capture/Compare regiszter a benne tárolt értékkel minden pillanatban összehasonlítja a számláló aktuális értékét, és ha egyezést talál, akkor egy Flag

[11] A számlálók órajele a két címbuszra való csatlakozástól függ: ugyanis az egyik busz 36 MHz-es, míg a másik a teljes sebességű átvitelre is képes, azaz 72 MHz-es.

generálódik, amelyre szintén megszakítás indítható. Ezzel a funkcióval lehet például a tengely forgatása során egy adott pozíciót detektálni, vagy meghatározott inkrementek közti tartományokat figyelni.

- *PWM generálás:* impulzusszélesség-modulált jelek előállítására alkalmas.
- *One-pulse mód:* elindítja a számlálót egy jel hatására, majd adott késleltetés után meghatározott mennyiségű impulzust generál.

Számunkra az első két működési mód a fontos, mert a szenzorok által szolgáltatott jelek elkapásához és különböző események keltette programmegszakításokhoz e módokat kell alkalmazni.

Az egyik legfontosabb mód, amiről eddig nem esett szó, az vezetett ennek a típusú processzorcsaládnak a kiválasztásához. Ez pedig az időzítő egységek *Encoder Interface* módja. Ebben az üzemmódban a timer négy csatornájából kettő csak az enkóderes jelek fogadására figyel. Ilyenkor a számláló tulajdonképpen úgy viselkedik, mint egy szimpla külső órajel, amely beépített irányváltási logikával rendelkezik. Ez azt jelenti, hogy a számláló csak a zérus kezdeti érték és az előre beprogramozott végérték között növeli, illetve csökkenti a tartalmát a forgásirány-detektálásra képes egységre támaszkodva. A detektáló modul egy fáziskülönbség érzékelő, amely a két csatorna minden egyes élváltásánál megvizsgálja a fázishelyzetet, és ennek megfelelően továbbít információt a forgatás irányáról (DIR, azaz direction bit) az inkrementálást/dekrementálást végző modulnak. A számlálás iránya megfeleltethető az enkóder forgási irányának, a számláló tartalma pedig az enkóder pozíciójának. Encoder Interface mód esetén a korábban ismertetett funkciók (mint a Capture, Compare, Prescaler) ugyanúgy üzemelnek, és változtathatóak, mint normális működés esetén. Az enkóder differenciális kimeneteiről érkező jelek egy, az MCU-n belül található komparátoros egységen mennek keresztül, ami feldolgozható digitális jelekké alakítja azokat. Minden csatorna bemenetén zajszűrő található, amely a környezetből felvehető zajok intenzitásától függően sokféle érzékenységi módra állítható be. A szűrés tulajdonképpen egy mintavételi tartomány kijelölését jelenti: minél mélyebb szintű szűrést szeretnénk elérni, annál több mintavételi esemény bekövetkezésére és érvényesítésére vár a szűrő. Így minél több a minta, annál biztosabb a helyes jelszintérzékelés. Az elmondottak alapján belátható, hogy egy külső inkrementális forgójeladó csatlakoztatása esetén nincs szükség

semmilyen külső illesztő logikára, hiszen a processzor fel van készítve az ilyen típusú jelek fogadására.

Az STM32F103 mikrokontrollerhez ingyenes fejlesztőkörnyezetet[12] ajánl a gyártó, amely tartalmazza a programozáshoz szükséges editor-t, debugger-t, ARM C/C++ compiler-t. A gyártó honlapjáról ingyenes mintaprogramok, tesztkódok is letölthetőek, valamint széleskörű segítségnyújtási lehetőségeket is biztosítanak tudásbázissal és fórumokkal, amelyeken felhasználók és programozók oszthatják meg véleményeiket, tapasztalataikat.

#### 3.2.3.2 Fejlesztőpanel

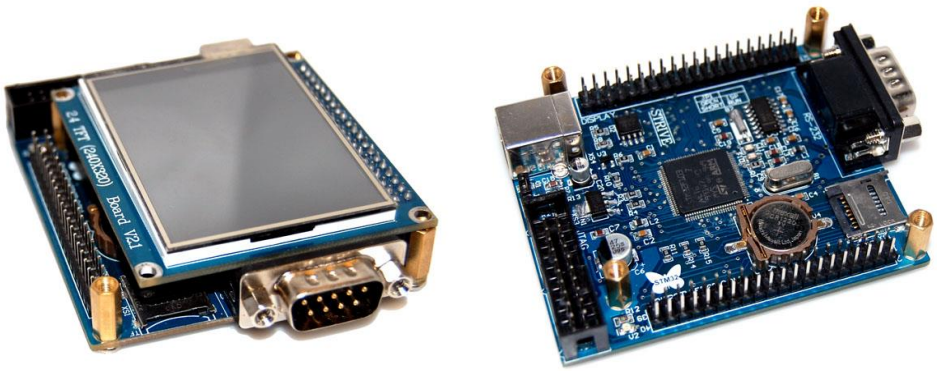
Maga a kontroller kiegészítő áramkörök nélkül nem használható semmire, ezért a feladatom megoldásához egy fejlesztőpanelra volt szükségem. Saját nyomtatott áramköri panel tervezése és legyártatása nagyon hosszú időt vett volna igénybe, ezért olyan megoldást kellett keresnem, amely a piacon már kész beágyazott rendszerként van jelen. A Mini STM32 DevBoard (lásd 18. ábra) egy olyan több elemből álló konstrukció, amely a felmerülő igényeket teljes mértékben képes kielégíteni. STM32F103 maggal rendelkezik, amelyhez az alábbi kisegítő egységek csatlakoznak, két panelen:

- tápellátás visszajelző LED
- funkcionális nyomógomb
- RS232 port PC-vel történő kommunikációhoz
- USB 2.0 a tápellátás biztosításához
- JTAG interfész a felprogramozáshoz és a debuggoláshoz
- 2 MB-os soros FLASH memória
- Micro SD foglalat
- gombelem aljzat
- egy 2,4 hüvelykes TFT (240x320) LCD kijelző 16 bites vezérlő logikával.

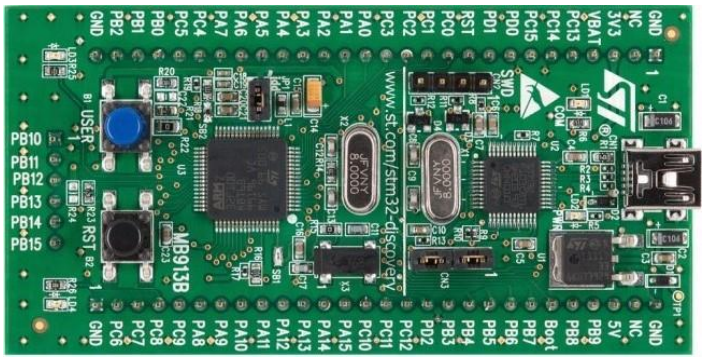
A panel JTAG modulja a mikrokontroller felprogramozását, és a szoftverben történő hibakeresést (debug) segíti, viszont a PC-vel történő kommunikációhoz és a programozási művelethez szükséges különböző konverziókhoz egy további elemre volt szükség. Erre a célra a 19. ábrán látható STM32LVDISCOVERY fejlesztőkártyát

[12] Keil µVision: http://www.keil.com/arm/mdk.asp?gclid=CPDi-PnmqroCFc1V3godDV4ABg

választottam, amely igaz, hogy egy újabb hardvert visz a rendszerbe, de ez a legegyszerűbb és legolcsóbb megoldás, amellyel az STM32F103 kontrollert fel lehet programozni. Az információkat a fejlesztőkörnyezettől Mini-B USB-n keresztül kapja meg, majd a megfelelő átalakításokat követően ST-Link protokoll használatával és szalagkábel segítségével hajtja végre a tényleges fejlesztőkártya flash-elését.



**18. ábra: Mini STM32 DevBoard**




**19. ábra: STM32VLDISCOVERY kártya**

## 3.3 Összefoglalás

A harmadik fejezet ismertette a fejlesztési folyamat kezdeti fázisaiban megfogalmazódó követelményeket, végül beépítésre kerülő eszközök tulajdonságait, belső felépítését, működési módjait, valamint azt, hogy hogyan is képesek eleget tenni a követelményeknek. A rendszerterv elkészítése és az alkatrészek beszerzése után következett a megvalósítás fázisa, amely a negyedik fejezetben kerül bemutatásra.

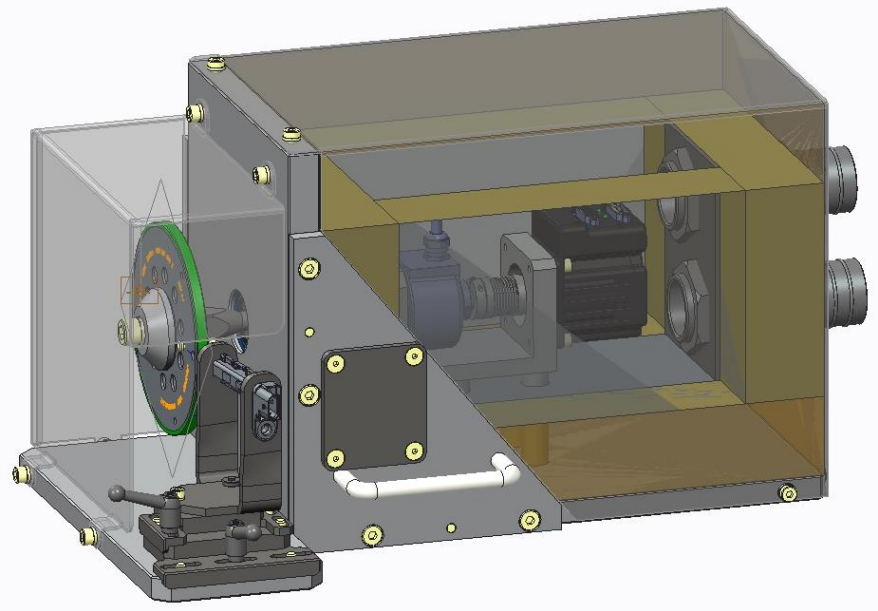
# 4 Megvalósítás

Ez a fejezet az elkészült berendezést mutatja be, hogy milyen hardveres és szoftveres komponensekből épül fel, és azok hogyan működnek külön-külön, illetve együtt, egész rendszert alkotva.

## 4.1 Mechanika

A mechanikai egység egy hőálló fém házból és a benne található szervomotoros hajtásból áll. Az igények és követelmények megfogalmazása után a háromdimenziós tervek elkészítését és a legyártást egy külső cég végezte. A Hardver alfejezet a házat mutatja be kívül-belül, a 4.1.2 alfejezet pedig a szervomotor felprogramozásáról és működtetéséről szól.

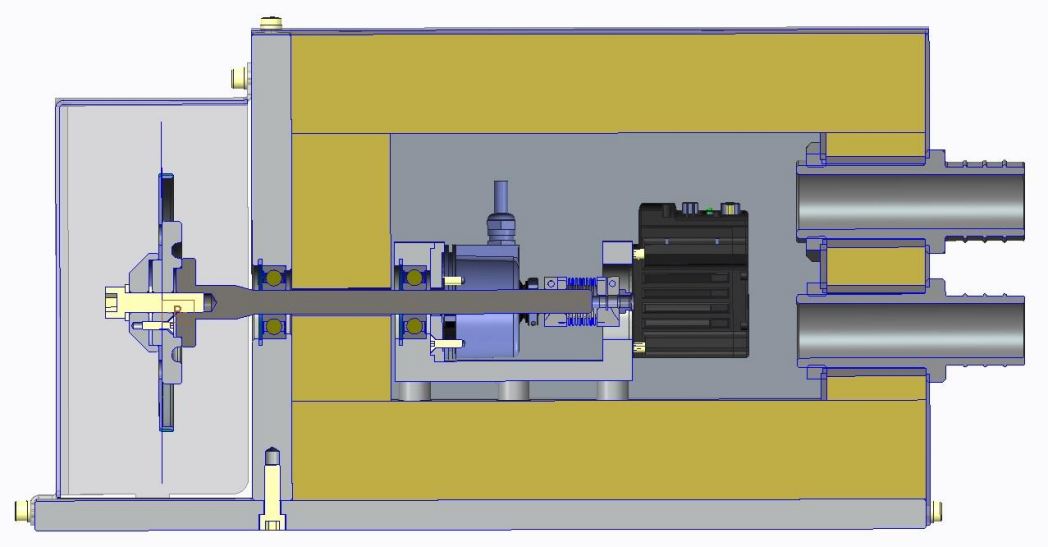
### 4.1.1 Hardver



**20. ábra: Szenzorvizsgáló külső nézetben**

A szerkezet – ahogy a 20. ábrán is látható – kívülről egy egyszerű fémdoboznak tűnhet, amelynek elülső feléből egy tengely lett kivezetve. Ennél azonban jóval többről van szó. Mivel az eszközt klímakamrában szeretnénk használni, ezért olyan anyagokra és megvalósításokra volt szükség, amelyek ellenállóak a hőingadozással, páratartalommal, korrodálódással, mechanikai behatásokkal szemben. A fém borítás védelmi funkciókat lát el (hő, ütés, korrózió elleni védelem), amely alatt vastag

szigetelő réteg található. A szigetelés teljesen beburkolja a belső teret, csökkentve a külső környezeti hőmérséklet által esetlegesen okozott káros behatásokat.



**21. ábra: Szenzorvizsgáló metszeti képe**[13]

Belül a szervomotoron túl helyet kapott még a hozzá kapcsolódó tengely és egy tengelykapcsoló. Mivel a forgatandó tengely végét leterheljük különböző nehézségű tárcsákkal, ezért ilyenkor a tengely hosszából adódó erőkar túl nagy radiális irányú nyomatékot vihet a rendszerbe. Ezt a hatást csökkenti a megfelelő csapágyazás és a tengelykapcsoló, továbbá a kapcsoló biztosítja, hogy a meghosszabbításként funkcionáló tengely végén a motor által leadott nyomaték ugyanakkora maradjon, mint közvetlenül a motor saját tengelyére leadott nyomaték. A SmartMotor és a tengelykapcsoló belül olajos kenéssel és csapágyazással lett kezelve, amik csak bizonyos hőmérsékleti tartományokban látják el a feladatukat. A túl meleg a nagy hőtágulás miatt tönkreteheti a motorban található beágyazott logikát is, a túl hideg pedig a kenőanyag vetemedését okozhatja, ami alkatrészek közti súrlódáshoz vezethet. Ezért mindenképp gondolni kellett valamilyen hűtési/fűtési funkcióra is. A ház hátsó felén két nagy átmérőjű nyílás található. A hozzájuk csatlakoztatott csöveken keresztül lehet a belső térbe juttatni a tápellátáshoz és kommunikációhoz szükséges vezetékeket, valamint a klímakamrán kívül elhelyezett ventilátor segítségével légkeveréses hűtés és fűtés biztosítható, amely nem engedi, hogy a hőmérséklet a motor környezetében

[13] Megjegyzés: a tervrajzon a tengelykapcsoló és az első oldali borítólemez között látható egy plusz alkatrész, amely még nem került beépítésre. Ez egy 300 ezer felbontású inkrementális enkóder, amely a jövőben a nagyobb pontosság elérése érdekében kiváltja a SmartMotor enkóderét. Részletesebben az 5. fejezet ír erről az eszközről.

+100 °C fölé vagy -40 °C alá essen. A ház oldalán fogantyúk is helyet kaptak a könnyű szállíthatóság érdekében. Az egyik fogantyú fölött egy fekete színű, vasból készült rész látható, amelyre kalibrálási és finommérési célokból mágneses rögzítéssel mérőóra helyezhető el.

A ház elülső oldalán, a tengely kivezetésénél zajlik a tényleges mérés. A tengelyre az összes típusú tárcsa azonos illesztési móddal csatlakoztatható. A rögzítés egyszerű csavarozással lehetséges, a menetes furat a tengely közepében végződik.

A mérési pontosság lelkét a rendszer kiegyensúlyozása adja. Minden egyes tárcsát és magát a tengelyt is precíziós méréssel állítottak be. A tárcsáknál a fő szempont az volt, hogy a felfogató papucsok pont a tárcsák közepére kerüljenek, valamint, hogy a különböző fogak által keltett forgás közbeni ütések minél kisebb mértékben hassanak a tengelyre. A papucsokba körkörösen furatokat fúrtak, amelyekbe kiegyensúlyozó csavarokat helyezve történt meg a kiegyensúlyozás. A tengely végén található csatlakozási ponton szintén ugyanilyen furatok és csavarok kaptak helyet. A kiegyensúlyozás végén a csavarokat beragasztották, hogy azok a felhelyezések és levételek során ne mozdulhassanak el.

A szenzorok pozícionálását kézi keresztszánnal lehet elvégezni. A két fogantyú segítségével külön-külön oldhatók és rögzíthetők az előre-hátra, illetve oldalirányú mozgásra képes szánok. Így állítható be a szenzor és a tárcsa egymáshoz képesti távolsága (azaz a légrés nagysága), illetve ha szükséges, egy síkba lehet őket hozni (a szenzorok magasságát nem kell változtatni, mivel minden egyes szenzorrögzítő azonos magasságú). Mivel minden szenzor más alakú, ezért mindegyikhez egyedi felfogató került legyártásra. A felfogatók a keresztszánon megvezetés segítségével csúsznak a pontos helyükre, ahol csavarral rögzíthetőek. Ha új mérést szeretnénk indítani egy másik típusú szenzorral, akkor csak ki kell csavarozni a tengelyből és a keresztszánból a rögzítőket, levenni a tárcsát és a szenzortartót, majd az újakat a helyükre rakni. A művelet gyorsan és egyszerűen elvégezhető, és nem kell megbontani az egész rendszert, ha változtatni szeretnénk a mérési elrendezésen.

Még egy elemről nem esett szó, amely szintén a ház elülső oldalán található. A fel-, és lehajtható rácsos szerkezet baleset-megelőzési célból került felhelyezésre. Véd a tárcsa lelazulása és elrepülése esetén, valamint gondoskodik arról, hogy működés közben senki se érintkezhessen a gyorsan forgó alkatrészekkel.

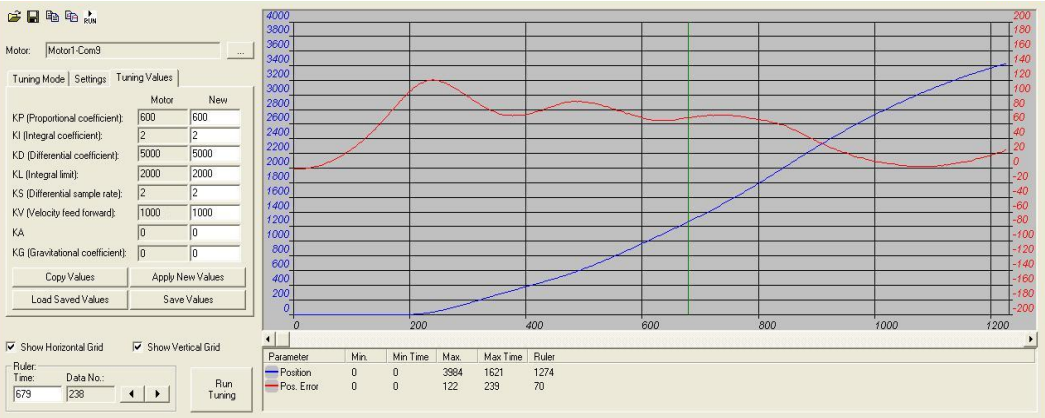
### 4.1.2 Motor és enkóder

A SmartMotor SM23165D egy intelligens szervomotor, amely saját programnyelvvel és fejlesztőkörnyezettel rendelkezik. Programnyelve hasonlít az Assembly-re és a Basic-re, viszont a mozgatásokhoz és szabályozásokhoz specifikus utasításkészlete van. A fejlesztéshez egy összetett és rendkívül hasznos grafikus felületű programcsoportot biztosít a gyártó. A fő program, amelyre a többi funkcionális alprogram épül, az SMI (Smart Motor Interface). Segítségével kapcsolatot tudunk kialakítani a motorral, visszakövethetjük a kommunikációs vonalon futó adatcsomagokat, utasításokkal vezérelhetjük a motort, illetve programokat is írhatunk. További alprogramjai, amelyeket használtam a fejlesztés során:

- *SMI Playground*: olyan grafikus felület, ahol egérrel, egyetlen kódsor írása nélkül is irányíthatjuk a motort. Hasznos, ha különböző mozgási profilokat szeretnénk kitalálni, tesztelni és mindezt gyorsan, programozás nélkül.
- *SMI Motor View*: az épp csatlakoztatott motorunkról kaphatunk hasznos információkat (mint például az aktuális fordulatszám, pozíció, hőmérséklet, flag-ek értékei és különböző hibajelző bitek állapotai), és átváltásokat végezhetünk a programváltozók és a valóságos fizikai paraméterek között. Utóbbira azért van szükség, mert a motor kontrollerében az adott fizikai tényező (fordulatszám, gyorsulás, stb.) arányos kapcsolatban áll a hozzá definiált változóval, amit át kell számolni a motorban található mikrokontroller által fogadható információvá (például egy 100 fordulat/perc-es fordulatszám MVT = 54600 változóértéknek felel meg).
- *SMI Tuning Engine*: a motor a szabályozást bonyolult szervo, elektronikus fékrendszer és PID szabályozó segítségével valósítja meg. Ha valamilyen mozgás történik, akkor a szervo minden esetben bekapcsol. A mozgás befejeztével is tartja a pozíciót mindaddig, amíg nem kívánunk mást tenni. Ha megpróbáljuk kézzel kitéríteni a motor tengelyét a stabil helyzetéből, akkor szabályozással próbál visszatérni az adott pontba. Ez nagyon hasznos funkció, hiszen így nem kell tartani attól, hogy megállítást követően a tengely véletlen hozzáérés hatására elfordul, és emiatt a számított értékek „elmásznak”. Viszont hátulütője is van, mert mindig szoftveresen ki kell kapcsolni a szervo-t, ha bármilyen változtatást kívánunk alkalmazni a tengelyen (például ha egy tárcsát

le szeretnénk cserélni, akkor a csavarok lazítása és feszítése károsíthatja a szervo-t, ami kerülendő).

Mivel minden tárcsa más tömegű, ezért máshogy terhelik a motort és a szervo-t is. A szervo, ha még nem lett módosítva, akkor a gyárilag kalibrált PID paraméterek szerint működik. Ez viszont nem elégséges különféle nehézségű tárcsák forgatásakor, mert a terheléstől függően máshogy fog viselkedni a motor. Ha nem az adott értékeknek megfelelő tömeget kívánjuk megforgatni, akkor a szabályozó próbálja az optimálishoz közelíteni a nyomatékot, és ezáltal a mozgást, viszont ez sokszor gerjedéshez, oszcillációhoz vezet, ami daráló, hirtelen oda-vissza forgást eredményez. Ha el szeretnénk kerülni ezt a helyzetet, akkor a Tuning Engine-t kell használni. A program lehetőség nyújt az összes szabályozási paraméter külön-külön történő változtatására, hibahatárok és jelző limitek beállítására, a gravitációs erő, a gyorsítási nyomaték és a PID-hurok késleltetésének figyelembevételére. Az SMI Tuning Engine tehát nagyon összetett program, amellyel egyszerűen létre lehet hozni az adott terheléshez a tökéletes forgatási profilt. A cél az volt a beállítások során, hogy olyan paraméterezést találjunk, aminél a motornak a legkevésbé kell beavatkoznia, ugyanis így a legkisebb a valószínűsége az oszcillációnak. Ezt a folyamatot minden egyes tárcsára külön-külön el kellett végezni. A 22. ábrán a vezérműtengely jeladó PID paraméterezése látható.




**22. ábra: Vezérműtengely tárcsa PID paraméterezése**

A piros, hullámosabb jelalak jelenti a pozícióhibát, a kék pedig a célt, ahova a motor tengelyét el szeretnénk juttatni. Máshogy fogalmazva a piros görbe a szabályozó beavatkozásának mértékét jelzi, amellyel a kék görbét minél rövidebb idő alatt és minél pontosabban lineárishoz tudja közelíteni. A kép bal

oldalán a számértékekkel megadható paraméterek láthatóak, melyekkel a beavatkozó működését, beleszólásának mértékét lehet változtatni. A PID értékek beállítása során egy olyan problémával találkoztam, amely a teljes beállítási művelet újragondolását eredményezte. Először a paramétereket 25 VDC táplálás mellett állítottam be és furcsállva tapasztaltam, hogy hiába történt meg a „tuningolás", 48 VDC maximális tápfeszültség mellett a motor nem az elvártnak megfelelően viselkedik, hanem az összes tárcsa esetén oszcillálni kezd. A jelenség annak tudható be, hogy a motor különböző táplálás esetén eltérő értékű nyomatékot képes leadni a tengelyére. Amikor nagyobb feszültséget kapcsolunk rá, akkor nagyobb nyomatékkal próbálja a kis nyomatékra kalibrált PID szabályozást megvalósítani, ami gerjedéshez vezethet. A probléma megoldása az volt, hogy a paraméterek beállítását az összes tárcsánál a legnagyobb rákapcsolható feszültségen kellett elvégezni, amely kisebb nyomaték esetén is működőképes mozgást eredményezett.

Az SMI fejlesztő program ismerete az analízis-berendezés felhasználóitól nem várható el, ezért egy olyan megoldásra volt szükség, amely kevés művelet elvégzését követően a kívánt forgási profilt automatikusan a motorba tölti. Ehhez a SmartMotor működtetését LabView környezetbe ágyaztam speciális bővítmény segítségével. A motort kétféleképp lehet üzemeltetni: egyikben adatokat fogad a perifériáján és a rá fordított program dolgozza föl azokat, másik módban pedig az adatok vezérlőszavak, amelyek a működését befolyásolják. A mérőberendezésnél a motor a második üzemmódban működik és a LabView felület (melyet a 4.2.2 pontban részletesebben ismertetek), soros porton keresztül utasításokat küld a motornak, amelyek alapján az elvégzi a megfelelő feladatokat.

A motornak a táplálást egy 0…84 VDC 10A labor tápegység biztosítja, a kommunikáció soros porton zajlik. Az enkóder csatornái a motor egyes kimeneteire közvetlenül ki vannak vezetve, amelyek árnyékolt kábellel csatlakoznak a fejlesztőpanelhez.

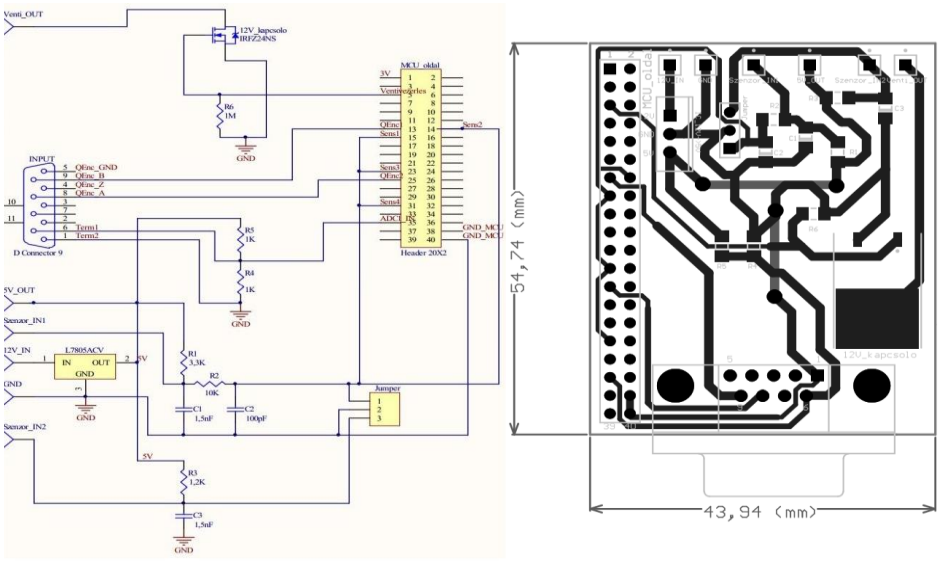
## 4.2 Többfunkciós nyomtatott áramkör

A követelményeknél a szenzorok vizsgálatával kapcsolatban az volt a kritérium, hogy egy olyan jelillesztő elektronikai panelra van szükség, amely az 5 V-os jelszintet a mikrokontroller által elfogadható 3,3 V-osra redukálja és emellett a nagyfrekvenciás parazita zavarjeleket kiszűri. A fejlesztés során viszont további igények is felmerültek, melyek a nyomtatott huzalozású lemez funkcionális bővítését idézték elő.

Az egyik fontos paraméter, melyről mindenképp szeretnénk informálódni, a fém gépházon belül uralkodó hőmérséklet. Erről valamilyen visszajelzésre van szükség, hiszen fontos tudni, hogy a hűtőventilátort mikor kell be-, illetve kikapcsolni.
A fejlesztés fázisában a jelillesztés próbapanellal és furatszerelt ellenállásokkal, kapacitásokkal történt, amelynek nagy hátránya, hogy könnyen szétcsúszhat, kijöhetnek belőle az egyes elemek és sok kábelt igényel. Ennek orvoslására mindenképp ki kellett találni egy olyan megoldást, amellyel nyugodtan felhasználó kezébe bocsátható a rendszer. A további cél tehát az volt, hogy a teljes mikrokontrolleres egység és a hozzá kapcsolódó vezetékezés egy műszerdobozon belül elférjen. Ezzel megszűnhet a bonyolult vezetékhálózat és egy esztétikus, letisztult architektúrát kapunk eredményül.

A 23. ábrán látható a kapcsolási rajz és a panel huzalozása. A táplálást 12 V-os egyenfeszültségű labortápról kapja, amelyet egy L7805ACV feszültségszabályozó 5 V-ra redukál. Ezt az 5 V-ot kapják meg a panelon található ellenállások és kondenzátorok, továbbá maga a szenzor is az 5V_OUT pad-ről. A panelon két 5 V-ról 3,3 V-ra történő jelillesztés is lehetséges, egyik a vezérműtengely, másik a multipolrad főtengely jeladó katalógusában megadott kapcsolás alapján. A két lehetőség közül egy három tüskesoros Jumper segítségével lehet választani. A második illesztés nem szükségszerű, mert a mérések során a két ellenállásból és két kapacitásból álló logika (vezérműtengely jeladó dokumentációjából) az összes szenzorra kifogástalanul működött, egyszerűen opcionális jelleget képvisel. A DB9 csatlakozón érkeznek a kvadratúra enkóder csatornái, valamint a hőmérsékletérzékelésre alkalmazott termisztor két jele. Egy párhuzamosan kapcsolt 1 kohm-os ellenállás segítségével alakítja az ohmikus mennyiséget feszültségjellé, amely a mikrokontroller A/D átalakítójának bemenetére kerül. A 40 pin-es tüskesor a Mini STM32 DevBoard-ra csatlakozik és az STM32F103 perifériáival köti össze a nyomtatott huzalozású panelt, és rajta keresztül az enkóder és a vizsgált szenzor jeleit. A ventilátor a kontrollerből vezérelve kapcsolható a következő folyamattal: ha a mért hőmérséklet túl magas, az MCU egy

GPIO portot 3,3 V-ra húz fel, amely egy IRFZ24NS MOSFET Gate-jére kerül, és a FET a kapcsolóüzemű működésének köszönhetően a Venti_OUT pin-en keresztül beindítja a ventilátort.




**23. ábra: NYÁK kapcsolási rajza és PCB képe**

A NYÁK tervezését az Altium Designer Studio programban végeztem, a huzalozást nyomtatásos, vasalásos és maratásos technológiával vittük fel a hordozóra. Az öt bemeneti és egy kimeneti pad-re vezeték lett forrasztva, melynek másik vége a műszerdoboz egyes banánhüvelyeire csatlakozik.

## 4.3 Szoftver

A fejlesztés leghosszabb ideig tartó és legkomplexebb feladata a programkódok megírása volt. Kezdetben csak a mikrokontrollerre fejlesztettem és a számítások eredményeit a kijelzőn jelenítettem meg. Később, amikor már szükségszerűvé vált a nagyobb adathalmazok kezelése és figyelése, akkor a beágyazott rendszerre történő kódolással párhuzamosan bekapcsolódott a LabView környezetben történő felhasználói interfész kialakítása. Az alábbi pontok ismertetik a két, egymástól teljesen különböző, mégis egymással szoros kapcsolatban álló program felépítését és működését.

### 4.3.1 MCU

A mikrokontrolleres egység képezi a rendszer magját, a legfontosabb jelfeldolgozási és számítási műveleteket ez végzi el. A szenzor kimenetéről az 5 V-os nyers digitális jelek jelformálást követően 3,3 V körüli amplitúdóval érkeznek a Mini

STM32 DevBoard-ra. A panelen található 40 pin, melyekre a processzor belső egységeit (például timer-ek egyes csatornái), analóg jeleket, stb. ki lehet vezetni a Remapping[14] műveletnek köszönhetően. A szenzorok és a kvadratúra enkóder jelei digitális négyszögjelek, felfutó és lefutó élek egymás utáni sorozatai. Ezekhez olyan komparátoros logikára van szükség, amely képes megkülönböztetni a jelváltozás irányát. Ilyen logika például egy számláló/időzítő egység, amelyből összesen 14 található a kontrollerben, ami bőven elegendő a feladatok megoldásához.

A kiszámítandó értékek, melyeket mindenképp szeretnénk tudni, a következőek:

- *fordulatszám:* a tengely percenként megtett körülfordulásainak száma.
- *szögelfordulás:* a tárcsán egy mechanikai fog (vagy lyuk) elfordulásának szöge, amelyet a szenzor érzékelt. Ebből az értékből és a kvadratúra enkóder által szolgáltatott referenciajelekből a szenzor *abszolút és ismétlési pontosság*a meghatározható.
- *impulzus időtartam:* a 2.2.3 pontban részletezett multipolrad típusú főtengely jeladónál a forgásirány az impulzusszélesség értékéből derül ki. Ez pedig nem más, mint egy egymást követő felfutó és lefutó él érzékelése között eltelt időtartam nagysága.
- *impulzusok száma:* ugyancsak a multipolrad jeladóknál célszerű az egy körülfordulás alatt érzékelt impulzusszámot figyelni, hiszen ebből kiderül, hogy a szenzor mind az 58 élt érzékelte-e, avagy kevesebbet, illetve többet.

Mindezekhez több timer-re van szükség, összesen háromra:

- Az első timer két csatornáján a kvadratúra enkóder jeleit fogadja, másik kettő csatornáján pedig a vizsgált szenzor jelét (egyiken a felfutó, másikon a lefutó élt figyelve). Az enkóder „A” és „B” csatornái kerülnek a timer megfelelő bemeneteire, majd az egység eldönti a forgatás irányát a fáziskülönbség érzékelőjének segítségével. Ettől eltekintve a timer egy egyszerű számlálóként funkcionál, melynek periódusa az enkóder inkrementszám-1 értékéig tart, azaz

[14] Remapping: a processzor GPIO lábkiosztásának rugalmasságát biztosító funkció, amely lehetővé teszi, hogy egy adott GPIO portra ne csak egy eszközt (például timer-t) lehessen kivezetni, hanem szinte tetszés szerinti perifériaelérést engedjen meg. A dokumentáció [12] tartalmazza, hogy mely GPIO portra milyen áramköri elemek funkcióit lehet kötni.

3999-ig. Minden beérkező enkóderimpulzusra a forgásiránynak megfelelően eggyel növeli, vagy csökkenti a tartalmát. Amint eléri a maximumot, illetve lefelé számlálásnál a minimumot (nullát), túlcsordulás történik, ami a körülfordulás bekövetkeztét jelzi egy megszakítási flag bebillentésével. A másik két csatorna a szenzorimpulzusok felfutó és lefutó éleit figyeli, és amint egyet elkap, azonnal jelzőbitet billent, amire megszakítás generálható.

- A második timer szimpla órajelosztást végez, mely segítségével a fordulatszám számítható. A számláló a 36 MHz-es buszra csatlakozik. Ha a periódus 9999-ig tart és a prescaler értéke 0, akkor kaphatjuk meg a maximális órajel-frekvenciát, viszont a fordulatszám számításához minél nagyobb időtartamra van szükség (célszerűen másodpercre). Ehhez a programban a timer 4 Hz-es (0,25 sec) frekvencián számol, mert így kis számmal kell szorozni a másodperc, illetve a perc eléréséhez, valamint a frissítési gyakoriság is elég nagy ahhoz, hogy még nagy fordulatszámokat is könnyedén számítani lehessen (ehhez a periódust 9999-re, a prescaler-t pedig 1799-re kellett állítani). A timer túlcsorduláskor megszakítási flag-et generál, ezzel ütemezve a hozzá tartozó műveleteket, programrészleteket.
- A harmadik timer szintén két csatornán fogadja a szenzor jelét az impulzusszélesség meghatározásához. Azért van szükség erre a harmadik időzítő egységre is, mert igaz, hogy az első is két csatornáján fogadja a szenzor jeleit, ám teljesen más számlálási ciklust valósít meg. Ott a kvadratúra enkóder minimális és maximális inkrementszámának elérésekor történik a túlcsordulás, míg ebben az esetben időtartamot szeretnénk vizsgálni, méghozzá µsec-os nagyságrendűt, amelyhez teljesen más órajel és számítási mechanizmus szükséges. A harmadik timer Advanced-control típusú, ami a processzor legnagyobb sebességű, 72 MHz-es sínjére csatlakozik. Tehát órajele maximális, amire azért van szükség, mert a µsec frekvenciában MHz-es nagyságrendnek felel meg. Ezzel érhető el az impulzusszélességek számításánál a legnagyobb felbontás és pontosság. Mivel a két csatorna a vizsgált szenzor jelét kapja meg, ezért a felfutó és lefutó impulzusok elkapására megszakítás generálható, amely felhasználásával az impulzusszélesség számítható.

A konklúzió tehát az, hogy bemeneti jelek fogadásához összesen négy plusz kettő, azaz hat GPIO portra van szükség. Ezt szerencsére minden további nélkül meg lehet oldani Remapping alkalmazása nélkül, hiszen van két olyan timer a kontrollerben, amelynek minden egyes csatornája ki van vezetve a fejlesztőpanel tüskesorára. Így a GPIO portok felprogramozását és élesztését követően a panel hat olyan bemeneti csatornával rendelkezik, amely képes fogadni a szükséges mennyiségű jelet.

A következő alfejezetek részletezik, hogy az imént felsorolt értékek kiszámítása hogyan történik, milyen algoritmusok és módszerek vezetnek el a megoldásokhoz.

#### 4.3.1.1 Fordulatszám-számítás

A fordulatszám meghatározásához egyszerre két timer adataira is szükség van. Ismerni kell az enkóder aktuális pozícióját, a forgatás irányát, az eddig megtett fordulatok számát, és mindezen értékekkel a pontos ütemezésekhez 4 Hz-enként (0,25 sec) kell elvégezni a számítást. Az aktuális pozícióra azért van szükség, mert a 4 Hz-enkénti megszakítás nemcsak körülfordulásonként, hanem fordulatok közben is érvényre juthat. Ez azt jelenti, hogy több egész fordulat is lezajlik, és az új megszakítás ugyanolyan valószínűséggel eshet pont egy egész fordulat kezdő-, vagy végpozíciójába, mint két fordulat közé. A két fordulat közti értékhez pedig tudni kell az előző megszakításnál regisztrált enkóderpozícióhoz képesti elfordulás mértékét. A törtfordulat értékét ennek a távolságnak az enkóder maximális felbontásához viszonyított aránya adja. Előfordulhat olyan eset is, hogy az előző elkapott érték még túlcsordulás előtti, a következő pedig már az utáni, ami azt eredményezi, hogy új egész fordulat történt, de a törtfordulat értéke negatív szám lesz. Például: tört_előző 3850-nél lett elmentve, és felfelé számlálásnál 3999-nél túlcsordulás történt (ami egy egész fordulatot eredményezett), és a következő elkapás 160-nál érkezett. Így (`tört_új)-(tört_előző) = -3690` lesz, nem pedig a kettejük távolsága, ami `3999-3850+160 = 309`. A kód az ilyen lehetőségekre odafigyel, és nem engedi az ehhez hasonló hibás értékekkel történő számításokat.

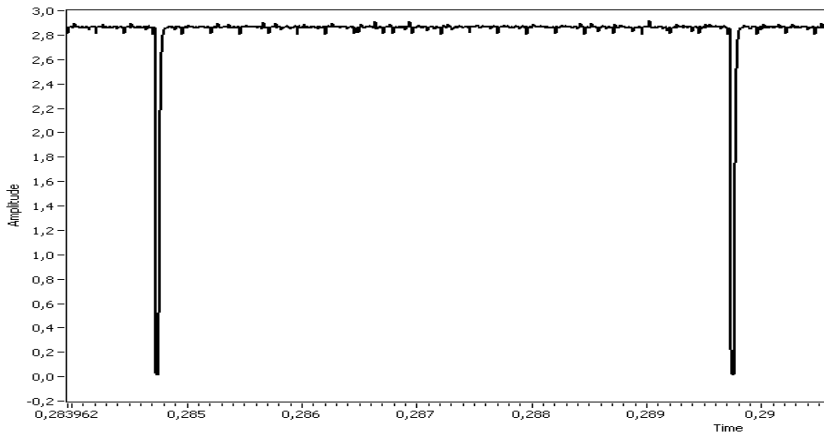
A timer az aktuális és az előtte lévő megszakításkor beolvasott és felhasznált adatokat elmenti, így az időközben megtett egész fordulatok számát könnyedén meg lehet határozni. Ehhez egyszerűen az aktuális és a negyed másodperccel őt megelőző egész fordulatok különbségét kell venni, és amint megvan az aktuális megszakításkor mind az egész, mind a törtfordulat értéke, a kettőt összeadva a negyed másodpercenként

meghatározott fordulatszámra jutunk. A helyes mértékegységhez az összeget 240-nel meg kell szorozni, és máris 1/perc-ben kapjuk a végleges eredményt.

A forgatás iránya minden egyes megszakításkezelő függvényben nagyon fontos paraméter, ugyanis egyik irányban az enkóder jelei felfelé számlálást váltanak ki, míg másik irányban lefelé számlálást. Mivel az enkóder inkrementszámára van szükség szinte minden paraméterhez, ezért az iránytól függően különböző számítási műveleteket kell végrehajtani. Ez a probléma először a fordulatszám meghatározásánál jelentkezett. Itt a törtfordulat használja az enkóder értékeit, és teljesen más az aktuális és az előző megszakításkor regisztrált inkrementek különbsége, ha felfelé, illetve lefelé számol az enkóderre specifikált timer számlálója. A felismerésnek köszönhetően a további modulok és interrupt rutinok is úgy lettek megírva, hogy a függvény kezdetén egy kiértékelés történik, amelynek eredményeként a végrehajtás a forgásiránytól függően csak az adott blokkon történik meg.

#### 4.3.1.2 Impulzus időtartam számítása

Az impulzusszélesség értékére, mint információra, a forgásirány-detektálásra képes szenzoroknál van szükség. Emlékeztetőül, a 2.2.3 pontban tárgyalt főtengely jeladó úgy jelzi az irányváltást, hogy dupla olyan hosszú ideig tart egy impulzus időtartama egyik irányban, mint a másikban, tehát minden beérkező impulzusról tudni kell, hogy időben milyen szélességű. Ezt a harmadik timer segítségével lehet meghatározni. A szenzorból kimenő jel két bemeneti csatornájára csatlakozik, egyik csatorna a felfutó élre érzékeny, másik pedig a lefutóra generál megszakítási jelzőbitet. Az élekhez a számláló aktuális tartalmát párosítjuk, majd az egymást követő le- és felfutó pulzusokhoz tartozó számlálóértékeket egymásból kivonva és 72-vel elosztva kapjuk µsec-ban az impulzusszélességet (azért 72-vel, mert a timer 72 MHz frekvencián működik). Ahogy azt a 24. ábra is mutatja, először mindig a lefutó él érkezik, majd utána a felfutó. Előfordulhat olyan eset, hogy a lefutó él még a számláló túlcsordulása előtt megérkezik, míg a felfutó már 0-ra váltást követően, de még nagyon kis értékkel. Ilyen esetben a különbség egy nagy negatív számot eredményezne, ám a program figyeli az ilyen eseteket is, és a számláló bitszámából adódó maximális értékének eggyel csökkentettjét egyszerűen hozzáadja a különbséghez (16 bit esetén $2^{16}$-1=65535-öt).




**24. ábra: Multipolrad jeladó impulzusai**

Hibás szenzorműködés adódhat rossz impulzusszélesség-értékekből is, ezért az összes szélességre minimum- és maximumkeresést is végrehajt a program, és kijelzi őket, amik alapján láthatóvá válik, ha hiba keletkezett.

#### 4.3.1.3 Szögelfordulás számítása

A mechanikai fogak, illetve lyukak szenzor által érzékelt szögelfordulásaira minden vizsgált jeladó esetén szükség van. Talán ez az a mért paraméter, amely a legkézenfekvőbb módon jelzi, ha hiba lép fel, viszont egyben a legösszetettebb programozási feladatot adta a szögek meghatározásában.

Köszönhetően a szimmetriának, a multipolrad jellegű tárcsáknál és szenzoroknál nem volt olyan nehéz az algoritmizálás, hiszen a szinkronimpulzus kivételével mindegyik azonosan 6° szögelfordulású. Külön megszakításkezelő rutinok foglalkoznak a fogak és a lyukak szögeivel. Egy fog a digitális jelben lefutó éltől felfutóig, míg egy lyuk felfutó éltől lefutó élig tart. Amint érzékelés történt, az adott élnél megszakítás történik, és az enkóder számlálójának értéke elmentődik egy változóba. A fel- és lefutásnál mentett számok különbségének és az enkóder maximális inkrementszámának hányadosa egy viszonyszámot ad, melyet 360-nal szorozva kapjuk a szögelfordulást fog és lyuk esetén is. Szemléletesen: `foghossz=((fel-le)/(maxinkrement-1))*360`.
A 4.3.1.2 pontban említett eset itt is felléphet, miszerint az egyik él még túlcsordulás előtt kerül regisztrálásra, míg a másik már átfordulás után. A program erre figyel, és elkerüli az ilyenkor adódó hibás eredményeket. A számított szögelfordulások minden értéke egymás után betöltődik egy tömbbe és a folyamat körülfordulásonként kezdődik elölről. Hibajelzésre több változót is használ a program. Amennyiben 58-nál kevesebb,

vagy több impulzus érkezett egy körbeforgás alatt, akkor egy-egy változó értéke növelődik (`hibaszámláló_több` illetve `hibaszámláló_kevesebb`).
A kiindulási (más néven szinkron) pozíciót is figyeli a rendszer. Ez azt jelenti, hogy a legelső fordulat után megjegyzi, hogy milyen enkóder értéknél jött az első 18°-os szinkronimpulzus, és minden további fordulatnál, amennyiben egy megadott szögintervallumon kívül esik a szinkron helye, akkor növel egy változót (`szinkronhiba`). Ezzel a három hibajelző paraméterrel nyomon követhető, hogy az összes fordulat során hány alkalommal jött több, kevesebb impulzus, illetve mennyiszer érzékelte rossz helyen a szinkronimpulzust a szenzor.

A programozás során a fő gondot kimondottan az eltérő szögelfordulású fogakkal és lyukakkal ellátott tárcsák okozták, ugyanis itt a forgatás irányától függően más-más sorrendben érkeznek a különböző nagyságú szögek. A fel- és lefutó élek érzékelése nem változik, ugyanúgy két megszakítási rutin figyeli külön-külön az élváltásokat.
A fogak és lyukak sorrendje, melyet a forgatás iránya szab meg, a hibák felderítése és jelzése szempontjából kiemelten fontos. A programot úgy írtam meg, hogy a tárcsát a mérés előtt mindig ugyanabból a pozícióból kelljen indítani. Ez egyéni megfontolásból és kényelmi célból valósult meg, ugyanis így csak egyetlen variációra kellett felkészíteni a kódot, nem az összes lehetségesre. Ha valamiért nem a kijelölt pozícióból indul a mérés, akkor más sorrendben érkeznek a fogak, ami a változók elcsúszását okozza. CCW forgatás esetén a fogak helyes sorrendje a 2.2.2 pontban részletezett tárcsánál, adott pozícióból történő indítás esetén 45° >> 6° >> 24° >> 12° >> 18°. Miután a program kiszámította a szögelfordulásokat, egy ötelemű tömbbe menti el őket és minden elemre elvégzi a katalógusban megadott tartományi ellenőrzést. Minden foghoz és lyukhoz egyedi intervallumok tartoznak, és ha a szenzor által érzékelt élváltásokból számított szög kívül esik az adott szögtartományon, akkor az adott indexű foghoz egy szintén ötelemű `foghiba` nevű tömb megfelelő indexű értéke eggyel növelődni fog. Ezzel nyomon követhető, hogy adott számú körülfordulás során az egyes fogakat (és lyukakat) hányszor érzékelte hibásan a szenzor. TPO módus esetén a szenzor adatlapjában megadott tűréshatárok mások, mint dinamikus működés esetén, ezért a TPO módus vizsgálatához külön hibaszámlálók kerültek definiálásra, és külön kiértékelő ciklus készült. A `foghiba_tpo` szintén egy ötelemű tömb, amely az adott indexű foghoz a megfelelő indexű elemét növeli, ha kilóg a számított érték a tűrésből. A

TPO vizsgálat az első hét felfutó impulzusig tart, amelyet egy változó figyel. Ha a változó értéke nagyobb, mint hét, akkor a program a dinamikus módhoz tartozó ellenőrzési ciklusra ugrik át.

Reprezentatív példa:

```
if(m<=7)
{
      if(((F[0] < 44.71) || (F[0] > 48.88)) && (f == 0))
      {
            foghiba_tpo[0]++;
      }
}
```

A legelső érzékelt fog a 45°-os, és a hozzá tartozó tartományok láthatóak a fenti kódban. Az `m` jelzi, hogy hányadik felfutó élnél tartunk, az `f` pedig biztosítja, hogy csak a nulladik indexű elem vizsgálata történhessen meg.

Ellenkező irányú forgatás esetén a fogak sorrendje máshogy alakul: 18° >> 12° >> 24° >> 6° >> 45°. A kód átszerkesztésével és az indexelés helyes átalakításával ez a forgatás is vizsgálhatóvá vált.

Az analógia a lyukak esetére is alkalmazható, csak ott természetesen más elnevezésű változókra van szükség. Továbbá olyan tárcsáknál is, ahol más elrendezéssel és más szögelfordulással ugyan, de ez a mechanikai fog-lyuk kialakítás létezik.

#### 4.3.1.4 Abszolút és ismétlési pontosság meghatározása

A főtengely jeladóhoz tartozó multipolrad tárcsán található egy bevágás, amely a fizikai referenciapont helyzetét jelzi. Ettől óramutató járásával ellentétes irányban forgatva 90°-nál helyezkedik el a szinkronimpulzus, amely utáni második impulzus lesz az a kitüntetett pozíció, ahol a gyártó méri az abszolút és ismétlési pontosságot. A TIM3 timer a TIM1 capture függvényén belül időzíti a lefutó élváltásokhoz tartozó TIM3 számláló értékeket, a felfutó élekhez tartozókat pedig a saját capture függvénye menti el (az okát lásd a 4.3.1.6 pontban). Ezek után minden fordulatnál egy vizsgálat következik arról, hogy az enkóder inkrementszáma éppen hol jár. Ha bal és jobb oldalról is beleesik a szinkron utáni második impulzushoz tartozó inkrementek tartományába, akkor belépünk a ciklusba, egyébként nem. A ciklusban egy százelemű segédtömb feltöltése történik az impulzus felfutó és lefutó éleinél elkapott inkrementekből képzett átlagértékekkel. Száz fordulat után a tömb már az összes információt tartalmazza, amely az abszolút pontosság számításához kell, ezért átadódik a `main` függvénynek és elölről indul a feltöltése.

Szemléletesen:

```
if(temp > min && temp < max)
{
      if(p_fel > p_le)
      {
            p_abszolut_seged[i] = (double)(p_fel - p_le)/2;
      }
      if(p_fel < p_le)
      {
            p_abszolut_seged[i] = (double)(p_fel-p_le+encmax)/2;
      }
      i++;
      if(i>=100)
      {
            i=0;
      }
}
```

A harmadik if ág a rendszer túlbiztosítása végett szükséges, hogy elkerüljük a nagy negatív eredményeket, amelyek a 4.3.1.1 pontban részletezettek alapján keletkezhetnek. A `main` függvényben, a pontosságok számításához megírt modulba minden századik fordulatot követően lép a program. Először összeadja a száz mintát a segédtömbből, majd az összeget elosztva százzal egy átlagértéket képez. Ezután a segédtömb minden eleméből keresi meg az átlagtól való legnagyobb eltérést, amely az abszolút pontosság értékét adja. Az ismétlési pontossághoz összegzi az összes eltérés négyzetét, ezt átlagolja (100-zal osztja), gyököt von belőle, végül a gyököt hárommal megszorozza, amely az ismétlési pontosságot adja eredményül. A kontroller tulajdonképpen programozással és tömbműveletekkel valósítja meg a 2.2.1.3 pontban megadott két képletet.

#### 4.3.1.5 Kommunikáció

A mikrokontrolleren futó program önmagában nem hasznos, ha nem informálja a felhasználót a mérések és számítások eredményeiről. Erre a célra két megoldás is született.

Az első a Mini STM32 DevBoard kijelzőjét használja az adatok közlésére. A képernyőre történő kiíratás a főprogramban (main függvényben) történik, mert így biztosítható a folyamatos és gyors megjelenítés. A kontroller LCD-vezérlést támogató fejlesztőcsomagja nagyon sokrétű kijelzésre ad módot. Az egyes karakterek adott pozícióba történő elhelyezése a vízszintes és függőleges koordináták megadásával lehetséges. Ezen felül választható betűszín, háttérszín és még sok egyéb olyan opció,

amely esztétikai célt szolgál. Minden szenzorhoz más-más adatok tartoznak, ezért a kijelző tartalma és elrendezése minden esetben eltérő. Vannak viszont állandó adatok, amelyek az összes szenzor esetén azonosan kiszámításra kerülnek. Ilyenek a fordulatszám, a megtett fordulatok száma, az enkóder inkrementszáma (tehát adott pozíciója). Ezek mindegyike megjelenik a kijelzőn, akármelyik szenzorhoz tartozó kód fut éppen a kontrolleren. Ezen felül minden egyes foghoz tartozó szögelfordulások, hibaszámlálók és impulzusszélességek (valós, minimum, maximum) is kiíratásra kerülnek az adott szenzorhoz optimalizált struktúrában.
A kijelzőre történő kiíratás a főprogramban történik az MCU teljes órajel-frekvenciáján, ezért minden adat a számítást követően szinte azonnal meg is jelenik. A soros kommunikáció során küldött és fogadott adatok helyessége pedig csak debug közben, vagy LabView környezetben ellenőrizhető. Ez azt jelenti, hogy ha egy adat valamiért hibás az USART vonalon, attól függetlenül a kijelzőn a mindenkori jó érték látható, tehát az LCD a fejlesztés közben fontos referencia szerepet töltött be a kontroller és a PC közti kommunikáció felépítésében. Az elkészült, működő rendszerben is ellátja ezt a szerepet, ugyanis ha valamilyen oknál fogva kételkedünk a LabView-ban jelzett adatokban, akkor még mindig ott van a kijelző, amely a soros átvitel hibáit nem tartalmazza, így összehasonlítva a két megjelenítés során közölt adatokat tudhatjuk, hogy a rendszerben van-e a hiba, vagy tényleg a szenzor érzékel rosszul.

A soros kommunikáció igénye már a fejlesztés korai fázisában felmerült. Célja, hogy:

- tehermentesítse a mikrokontrollert a felesleges és sokáig tartó folyamatok alól úgy, hogy a hosszabb műveletekhez szükséges adatokat átküldi egy sokkal nagyobb számítási kapacitással rendelkező PC-nek, amely ezek után könnyedén és gyorsan elvégzi a még hátralévő feladatokat.
- a felhasználó számára sokkal informatívabb tájékoztatási felületet lehessen kialakítani a LabView fejlesztő- és analíziskörnyezet segítségével.

Az adatok küldése előtt mindent egy `sendarray` nevű tömbbe tölt be a program. A továbbítás byte-onként történik, ezért minden egyes tömbelemre el kellett végezni egy számítást, hogy mekkora lehet a maximális mérete, majd ennek megfelelő számú byte-ra tördelve kerülnek be a változók egymás után a `sendarray` tömbbe. A tömb nem dinamikus memóriafoglalással van definiálva, mert az bármilyen meghibásodás

esetén túltelítheti a memóriát. Egyszerűen a program az elején lefoglal egy 190 elemszámú tömböt, amelybe biztosan belefér az összes széttördelt adat. Ez után a `main`-ben már csak meg kell adni, hogy az egyes szenzorok esetén mekkora indexig kell átküldeni a tömb elemeit. Ezzel a módszerrel minden szenzor esetén univerzálisan egyetlen tömbbe töltődik be az összes adat és az index jelzi, hogy adott típusú szenzornál mekkora lesz a tömb, és így mennyi elemet kell átküldeni. A szögelfordulás, impulzusszélesség, abszolút és ismétlési pontosságok esetén nem csak egész részt, hanem a tört részt is célszerű átküldeni, mert a tizedes jegyek is fontos információval bírnak. Ehhez az adott változót a kontrollerben felszorozzuk 10-zel, 100-zal, 1000-rel attól függően, hogy milyen pontosságot követelünk meg nála, aztán a felszorzott számot byte-okra tördeljük és egymás után átküldjük, majd a túloldalon fordított logikával visszaalakítjuk. A tördelés egy 2 byte-os adat esetén úgy történik, hogy első körben 0x00FF-el és 0xFF00-val logikai ÉS kapcsolatba hozzuk, amiből megkapjuk az alsó és a felső byte-ot külön-külön. A felsőt bitenkénti eltolással 8 bittel jobbra toljuk azért, hogy ne terheljük nagy méretű számokkal a soros vonalat (a PC oldalon $2^8$=256-tal beszorozva vissza is kapjuk a felső byte eredeti értékét), majd a felsőt, utána az alsót betöltjük a tömb megfelelő helyére. A `sendarray` tömb teljes feltöltése időzítve van, mert így a kontroller nincs folyamatosan lefoglalva, hanem csak időszakosan kell nagyobb műveleteket elvégeznie. A tömb minden körülfordulást követően egyszerre kapja meg az összes adatot. Az egyszerre átküldendő csomag kezdetének jelzéséhez egy ’@’ szinkronkarakter kerül kiküldésre, majd engedélyeződik az USART_TX vonal és a hozzá tartozó megszakítási rutin átküldi az adatokat a program elején megadott indexszámig. Ezt követően lezárja a küldési vonalat, letiljta a küldési megszakítást és törli a flag-et, visszaadva a műveletvégzést a fő programnak. Ezzel lezárul egy küldési ciklus és a következő körülfordulás után kezdődik elölről az egész. A szinkronkarakter kiküldése jelzi a LabView környezetnek, hogy új küldési ciklus indult és figyeljen, hogy mostantól kell az adott számú byte-okat fogadnia.

Kettő esetben adatok fogadására is figyelnie kell a programnak. Elsőként a LabView programjának elindulását követően alaphelyzetbe kell állítani a kontrollert, hogy minden változó felvegye kiindulási értékét és az LCD kijelzőre előzőleg kiírt adatok törlődjenek. A Reset igény megfogalmazódását követően a LabView egy karaktert küld az USART vonalon a beágyazott processzor felé, és az MCU ha ezt elkapta, akkor elvégzi az alaphelyzetbe állítási folyamatot.

Másik eset az, amikor a LabView a vizsgálandó szenzor kiválasztását követően tájékoztatni kívánja erről az MCU-t. A könnyű azonosításhoz minden szenzorhoz egy betűkód lett rendelve (A, B, C, D vagy E szenzor). A felhasználó jelzi igényét a LabView felé, hogy mely típus analízise következik, amit a megfelelő karakter soros porton átküldése és fogadása követ.
Mindkét fogadás (hasonlóan a küldésekhez) megszakítási rutinon keresztül történik. A beágyazott program legelején engedélyeződik az USART_RX vonal, amin keresztül a beérkező karakter a `menumode` nevű változóba töltődik be. A fogadásért felelő interrupt rutin nem tiltódik le, hogy bármilyen esetben, ha Reset-re lenne szükség, érvényre juthasson.

#### 4.3.1.6 A teljes program

A kontrolleren futó program egy sok modulból és elágazásból álló rendszer. Minden szenzorhoz egyedi programkód tartozik, de ahhoz, hogy moduláris, ám egységes rendszert hozhassunk létre, egy menürendszeren alapuló megoldásra van szükség. A menü segít eligazodni a sok szenzor miatt adódó választási lehetőségek között. A szisztéma olyan szempontból különleges, hogy a menü elemeit nem a kontrolleren tudjuk beállítani, hanem a LabView felületen keresztül kiválasztott mérési elrendezésnek megfelelő kód (A, B, C, D, E) alapján a kontrollernek küldött információból a blokkok közül csak bizonyos modulok futnak le. Összesen öt szenzor van, tehát végrehajtás szempontjából öt elágazás lehetséges a programban, viszont ha azt is figyelembe vesszük, hogy a forgatás iránya határozottan beleszól a beérkező adatokba és a számítási műveletekbe, akkor már öt szenzorhoz tíz lefutási lehetőség tartozik. Minden megszakítási ciklus a `menumode` változó kiértékelésével indul, amely előzőleg már megkapta a LabView-tól a vizsgálandó szenzort beazonosító betűkódot. Itt dől el, hogy a rutinokon belül mely modulok futhatnak le, és melyek azok a kódrészletek, amelyek már az adott szenzor szempontjából irrelevánsak.
A definiált változók univerzálisak, minden függvény és rutin hivatkozik rájuk. Változó tényező egyedül a `sendarray` tömb elemszáma, rajta kívül minden egyes tömb a legnagyobb előforduló mérettel lett definiálva, így kompatibilisek az összes szenzorra nézve. Például a vezérműtengely jeladó öt fog és öt lyuk mágneses jeleit érzékeli és továbbítja elektromos négyszögjelek formájában a kontrollernek. Viszont a 2.2.4 pontban, a 10. ábrán látható egy másik kialakítású, szintén vezérműtengelyre való

tárcsa, ami négy foggal és négy lyukkal rendelkezik. Egy ötelemű tömb lefoglalása esetén mindkét szenzor és tárcsa vizsgálható, a tömb feltöltése és kiolvasása szempontjából nem okoz problémát a két konstrukció különbözősége.

A program összesen három C fájlból áll.

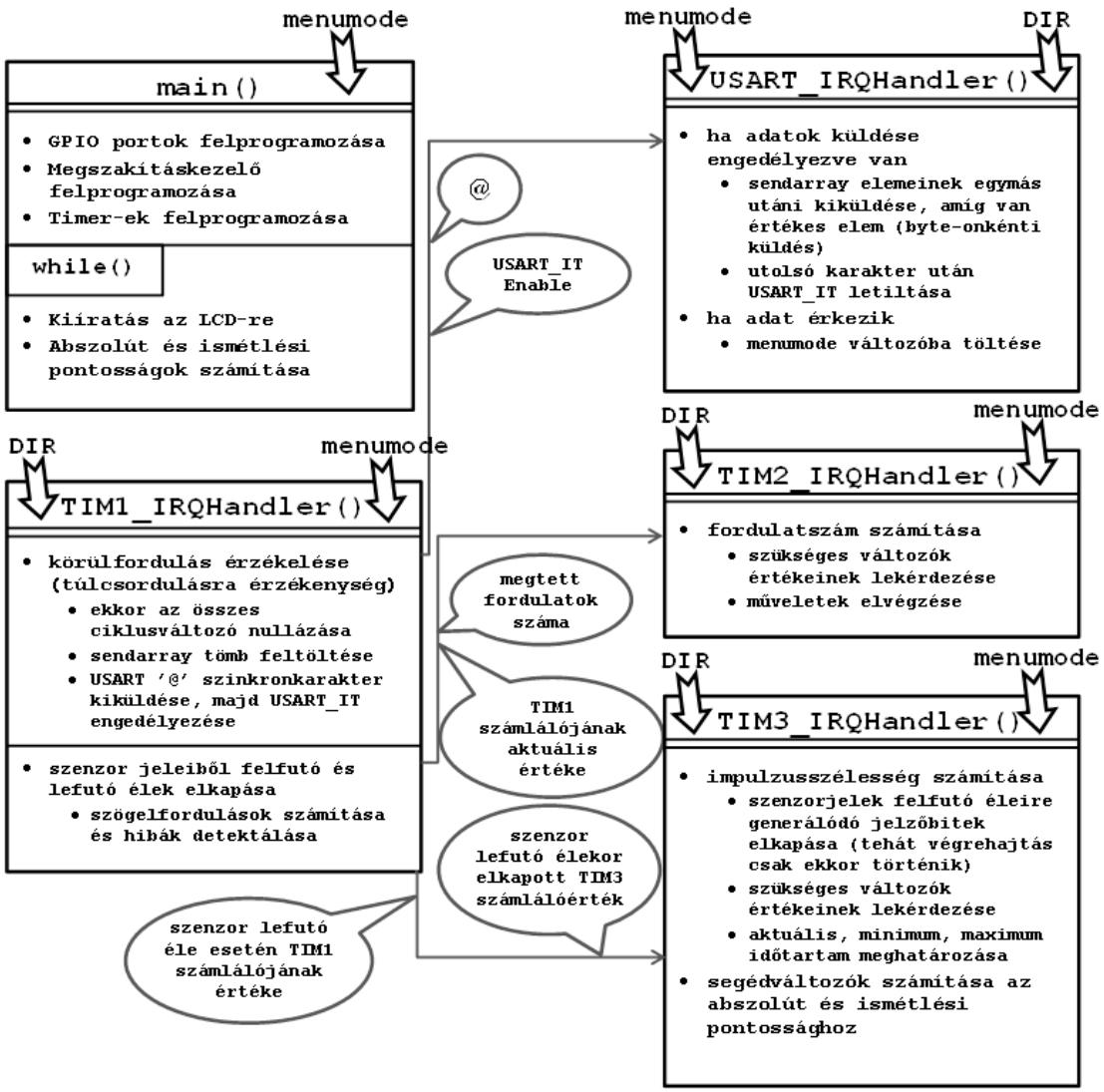

**25. ábra: Kontrolleren futó program elemei**

A main.c a program magja, minden analízis szempontjából fontos értékadás, függvényhívás itt történik. Ezen kívül az usart.c valósítja meg a maga négy függvényével a soros portra illesztéshez szükséges konverziókat és redukciókat, a button.c pedig a Mini STM32 DevBoard-on található egyetlen nyomógomb felhúzását és GPIO portra történő illesztését végzi el.

A fő program huszonöt függvényből, négy interrupt handler-ből[15], és a handler-eken belül összesen tizenkét elkapó modulból[16] áll. A handler-ek és a flag-ek változásainak észleléséért felelő rutinok egymás adatait is használhatják és használják, ezért a programozás során az ütemezésre és az összeakadástól mentes lefutásra külön figyelmet kellett fordítani. A 26. ábrán látható a teljes program moduláris blokkvázlata az öt fő egységgel, a köztük lezajló elérési utakkal és a lekért adatokkal, valamint a modulok belső felépítésére és elágazásaira legjobban ható `menumode` és `direction (DIR)` változókkal. Ahogy az ábra is illusztrálja, a `menumode` határozza meg, hogy az adott függvényen belül mely szenzorrészhez tartozó kód kiértékelésének kell megtörténnie. A `DIR` abban nyújt segítséget, hogy jelzi a forgatás irányát, aminek értékétől függően a forgásirányra érzékeny műveletek a megfelelő módban hajtódnak végre.

[15] Interrupt handler: megszakításkezelő függvény, amely a megszakítási eseményeket kiváltó egységhez (például timer egység által) tartozó folyamatokat magában foglalja és végrehajtja.

[16] Elkapó modul: olyan feltétel-elágazás, amely adott interrupt flag billenésére reagál és elvégzi a benne található feladatokat, míg más flag-ek aktivitását figyelmen kívül hagyja.




**26. ábra: Beágyazott program blokkvázlata**

Szinte minden rutin használja a többi rutinban található változókat is. Ez eseményvezérelt programozás esetén nehézségekhez vezethet, mert felléphet olyan eset, hogy egy megszakítási rutin olyan adat elérésére hivatkozik, amelyhez tartozó esemény még nem következett be, vagy már lezárult. Ekkor vagy nem éri még el a kívánt információt, vagy nem az aktuális van letárolva, hanem még egy előző megszakítás során keletkezett régi érték, amely az adott pillanatban már nem számít hasznosnak. Ezek az esetek hibás számításokhoz vezethetnek, ami ellen fel kell készíteni a rendszert.

Jelen esetben a védekezés úgy történt, hogy a már megírt interrupt rutinok kezdetéhez és lezárásához egy-egy analóg jelet küldtem ki egy GPIO portra, amelyet aztán oszcilloszkóppal figyeltem meg. Minden rutinnál ezt elvégezve az oszcilloszkóp négy csatornáján bizonyos hosszúságú négyszögjelek jelentek meg, amelyek az adott rutinhoz tartozó lefutási időtartamot jelezték. Az impulzusokból következtetni lehetett arra, hogy

melyik rutin az, amelyik túl sok ideig fut, illetve, hogy melyek azok, amelyek esetlegesen zavarhatják egymás elérését és működését. A megfigyelést alapul véve változtattam a kódon úgy, hogy a kritikus pontokban található (amik a leginkább lassították a rutinokat) változók értékadásainak ütemezését egy másik rutinba ágyaztam, így csökkentve a futási időt.

A 26. ábrán is látható az adatok útvonalaiból, hogy erre a módszerre az impulzusszélesség számításánál volt leginkább szükség. Ezt az indokolta, hogy a TIM3 saját capture csatornájának használata esetén a felfutó élek elkapása után nem azok a szélességek keletkeztek, amiket a katalógus említ, és amiket a szenzor működése alapján elvárnánk. Ezért van az, hogy a TIM3 nem a saját flag-elkapó rutinjában érzékeli a szenzorból jövő lefutó éleket, hanem a TIM1 ugyanezen esemény elkapására figyelő moduljánál menti el a TIM3 számlálójának akkori aktuális értékét. Mindkét esetben a szenzor lefutó éleit figyelik a rutinok, tehát ez az analízis során nem eredményez komoly eltéréseket, csupán az elérések ütemezését módosítja. Ezzel az ide-oda hivatkozással sikerült csak megvalósítani a TIM3-ban végrehajtandó számításokat úgy, hogy minden adat időben legyen elkapva, illetve felhasználva.

### 4.3.2 LabView

A programozási feladat másik nagy részét a felhasználói interfész kialakítása tette ki. Erre a legalkalmasabbnak a LabView fejlesztőkörnyezet minősült, mivel egy olyan programozási módszert kínál, amellyel nem csak grafikus felület hozható létre, hanem bonyolult számítási műveletek is megvalósíthatók, sőt akár egy valós mérőberendezés is lemodellezhető és szimulálható. Egy datalogger[17] segítségével például oszcilloszkóp által látott jelalakokat is „elkaphatunk” és megjeleníthetünk a virtuális kijelzőn, és az oszcilloszkóphoz hasonló műveleteket (zoom, kurzoros mérések) is végrehajthatunk a hullámformákon. Ezen túlmenően közvetlenül is rácsatlakozhatunk egy mérőberendezéssel a PC-re, amelyhez megfelelő könyvtári plug-in telepítése után a programon belül a PC-ben található processzor sebességével folytathatjuk az analizálást. Így gyorsabbá válhatnak a mérések, és nagyobb méretű adatokat is képesek vagyunk elmenteni. Összességében egy grafikus programnyelvet

[17] Datalogger: National Instruments által fejlesztett olyan külső hardveres egység, amely analóg és digitális jelek fogadására és rögzítésére alkalmas. Magas mintavételi frekvenciájának és belső memóriájának köszönhetően időben nagy tartományok vizsgálhatóak vele.

használó fejlesztőkörnyezetről beszélünk, amely tulajdonságait a gyártó honlapján található rövid, de annál informatívabb leírás jól összefoglalja, miszerint „a mérnökök és fejlesztők tapasztalattól függetlenül rövid idő alatt, költséghatékonyan hozhatnak létre illesztő felületeket a mérési és vezérlő hardverekhez, elemezhetik a mért adatokat, megoszthatják az eredményeket, és terjeszthetik a rendszereket” [13].

A LabView-ban írt program a beágyazotthoz hasonlóan nagy komplexitással bír. Egyszerre valósít meg tájékoztató, beavatkozó és archiváló funkciókat.

- *tájékoztat*, mert grafikus felületének köszönhetően a felhasználó értesülhet a mért paraméterek változásairól, figyelmeztető jelzéseket kaphat a fellépő hibákról.
- *beavatkozik,* mert irányítja a szervomotort és a mikrokontrollert, valamint befolyásolja a mérés befejezésének feltételeit.
- *archivál,* mert olyan többszintű regisztrálási rendszere van, melynek segítségével minden adat rögzíthető, minden készített hullámforma elmenthető és ezekből egy formanyomtatványt kialakítva további tájékoztatásra ad lehetőséget.

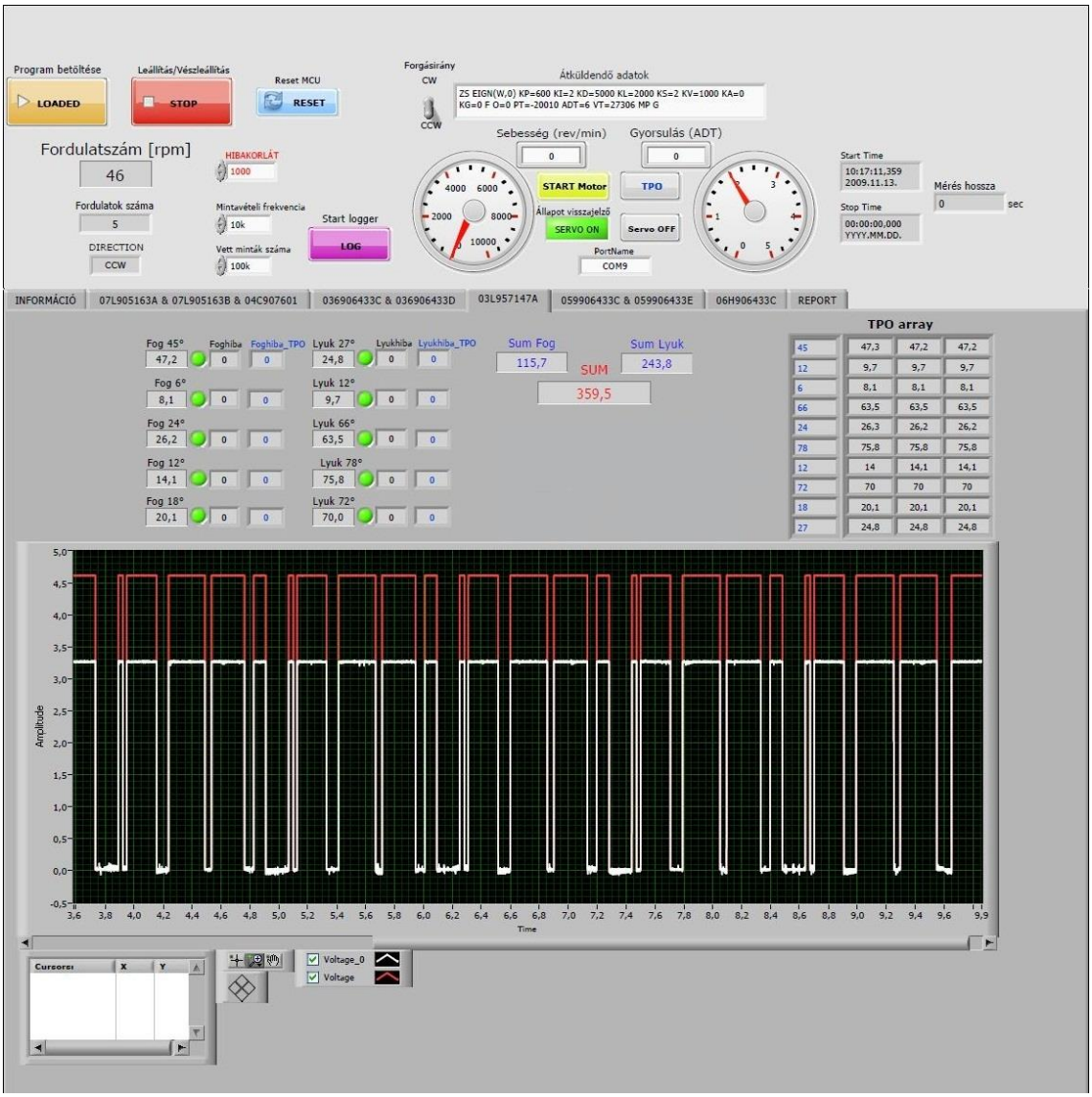
A LabView-ban történő fejlesztés nagy előnye (és természetesen nagyszintű bonyolultságának okozója is), hogy az összes megírt modul párhuzamosan és egyszerre hajtódik végre. Így gyorsan, nagy méretű adathalmazok vizsgálatára nyílik lehetőség, és a vizuális funkciók (mint például egy hullámforma megjelenítése) is valós időben elemezhetőek. A parallel lefutások hátránya a programozás során jelentkezik, ugyanis ha nem figyelünk oda egy ciklus leállítási feltételének ütemezésére, akkor egyes változókba betöltött értékek hamarabb vagy később érkeznek meg, valamint a következő szekvencia sem időben indul el vagy áll le. Ez akkor okozhat problémát, ha hirtelen szeretnénk valamit változtatni a programban (például leállítani valaminek a működését), ám annak indulását egy másik ciklus futása késlelteti. Az ilyen jellegű hibák megfelelő lokális változók használata esetén könnyedén elkerülhetők. A fejlesztés során én is arra törekedtem, hogy minden művelet a hozzá tartozó gomb megnyomását követően azonnal érvényre is jusson.

A programkód felépítését tekintve több részre osztható. Alapvetően minden analízishez kapcsolódó számítási művelet a mikrokontrollerből folyamatosan küldött, soros porton

érkező adatokra épül. A byte-ok beérkeztetéséért, a számítások elvégzéséért és utána az eredmények kijelzőre helyezéséért egy nagy while ciklus felel, melynek indulási feltétele nincs, minden programindítást követően el kezd futni. Amint a ciklus lezárult, a program egy egyszer végrehajtandó szekvenciára vált át. Ez először összefűzi az adatokat, majd egy jól strukturált jelentési ívet generál html vagy pdf formátumban. A program nagy részét e két blokk teszi ki. Rajtuk kívül még három while ciklus üzemel, melyek az MCU alaphelyzetbe állítását, a motorhoz kapcsolódó funkciógombok állapotának lekérdezését és a motor működtetését, valamint datalogger-en keresztül a különböző hullámformák megjelenítését végzik.

A programban minden változó egyszer van definiálva és lokális változók segítségével történik az értékük kiolvasása, módosítása. A változók lehetnek indikátorok vagy kontrollok. Az indikátor nevéből adódóan csak jelzési funkciókat lát el, tehát a benne tárolt adat kiolvasható, viszont a felhasználói kijelzőn futás közben nem módosítható. A kontroll változó segítségével lehet parancsokat kiadni, programelágazásokat vezérelni. Értékük variálható és tartalmuk hatással van a program további működésére. A program elején kezdeti értékadás történik, mellyel a lokális változókat alaphelyzetbe állítjuk. A lokális változók hasonlóak a C nyelvben a mutatókhoz. Értéket kaphatnak, de nem tárolják azt, hanem a definiált változóhoz hordozzák. Tulajdonképpen az eredeti változó olyan elérési lehetőségei, melyek nem terhelik le feleslegesen sok adattal a processzort, hanem csak hivatkozást jelentenek arra a változóra, amelyből képeztük őket.

A programfelület három részre osztható: indítási és leállítási műveletek, motorvezérlés és analíziskörnyezet. A 27. ábrán látható a program egy tipikus, analízis közbeni állapota. Bal oldalán találhatóak az indításért és leállításért felelős funkciók, tőlük jobbra középen a motorvezérlést és szabályozást végrehajtó modulok. A felület túlnyomó részét természetesen az analízishez kapcsolódó blokk teszi ki. Az alábbi pontok részletesen bemutatják, hogy mely nyomógombok mire valók és milyen műveletek elvégzését váltják ki.




**27. ábra: Vezérműtengely jeladó LabView vizsgálati felülete**

### 4.3.2.1 Indítási és leállítási műveletek

A program elindítását követően először mindig az információs felület jelenik meg, amely részletes tájékoztatást nyújt az összes nyomógomb, funkció és választható fül viselkedéséről és rendeltetéséről (lásd a Függelék 9.1 pontjában) Különböző instrukciók segítenek a program elindításában és a TPO mód használatában, mely pontokat követve biztosan sikeres mérést végezhet a felhasználó. Eközben a háttérben megtörténnek az inicializálások és a változók alaphelyzetbe állításai, valamint a mikrokontroller Reset-elése (amely manuálisan is elvégezhető a kék *RESET* gomb megnyomásával), amely után mindkét rendszer készen áll az analízis kiválasztására és elvégzésére. Ezt követően a program a fő while ciklusba lép és legelső lépésben kiad egy parancsot a motornak, hogy engedje el a szervóját. Ez lehetővé teszi, hogy a tárcsa

pozícionálását el lehessen végezni. Majd ki kell választani a fülek (*Tab Control*) segítségével, hogy mely szenzort szeretnénk vizsgálni, és a zöld *LOAD* gomb lenyomása ténylegesen elindítja az analízist. Ez a gomb mérésenként csak egyszer nyomható meg, ezzel biztosítva, hogy csak az adott szenzorhoz tartozó kód fusson a kontrolleren. Lenyomását követően a program elküld egy karaktert soros porton keresztül a mikrokontrollernek, amely a korábban már említett `menumode` változóba töltődik be. Ez a karakter beazonosítja a választott szenzort, betölti az LCD-re a hozzá kapcsolódó alapadatokat és kiválasztja a beágyazott programban azokat a részeket, amelyeknek a futtatására a mérés során szükség lesz. A LabView programjának belső felépítése olyan szempontból hasonlít az MCU-n futóra, hogy itt is minden művelet elején case struktúrájú programelágazás található, amely a kiválasztott szenzorhoz tartozó elágazási ciklusokat engedi futtatni, míg a többi eközben inaktív marad.

Mivel nagy sebességgel forgó alkatrészek is vannak a rendszerben, ezért a program megírásakor nagy figyelmet kellett szentelni a felhasználó biztonságára. Több művelet is a fő while leállításához, és ezzel egyetemben a motorfordulat nullára csökkentéséhez vezet. A kijelzőn lévő nagy piros *STOP* gombbal lehet az aktuális mérést befejezni, de vészleállító funkcióra is használható. Megnyomása után a program az összes ciklusból kilép és a motor minden esetben automatikusan megáll. A *HIBAKORLÁT* értékének beállításával megadható egy maximális szám, amelyet ha elér az összes figyelt hibaszámláló összege, akkor szintén minden folyamat és a motor is leáll. Ezt követően nem fejeződik be a program, hanem az analízis fázisából átkerül a riport felületre, melyről részletesebben a 4.3.2.3 fejezet szól.

A *LOAD* és a *STOP* gombok vezérlik a teljes analízist. Egy mérés csak a *LOAD* lenyomását követően tud elindulni, mivel addig a mikrokontroller sem csinál semmit, hanem csak vár az első értékes karakter beérkezésére. A *STOP* gomb lenyomása (vagy egy hiba esetén generálódó stop feltétel) pedig leállít mérést és motort egyaránt, és a *Tab Control* átugrik a *REPORT* fülre, amely jelzi, hogy vége a mérésnek, és már csak a dokumentálás van hátra. A *LOAD* és *STOP* gombok meghatározzák a mérés hosszát, ezért lenyomásukra időadat-rögzítés történik, és másodperc pontossággal elmentődik az aktuális rendszeridő ÉÉÉÉ.HH.NN. ÓÓ:PP:MM,MMMM formátumban. A befejezés időtartamából az indítását kivonva másodpercben kapjuk a teljes mérés hosszát, amely információ később hozzáfűződik a riporthoz is.

#### 4.3.2.2 Motorvezérlés

Ahogy a 4.1.2 pontban már említettem, a SmartMotor két működési módban használható. Adat módban a motorra küldött információkat adatként kezeli a beépített logika, míg kontroll módban mindent a motor vezérlésére hivatott kódként fogad. Ez teszi lehetővé, hogy LabView segítségével soros porton keresztül lehessen irányítani a motort különböző kódszavakkal. Erre egy példaprogram:

```
ZS EIGN(W,0)
KP = 600
KI = 2
KD = 5000
KL = 2000
KS = 2
KV = 1000
KA = 0
KG = 0
F
O = 0
PT = -20010
ADT = 6
VT = 27306
MP
G
```

Ez a kódrészlet egy TPO inicializálást végez, majd az eredeti pozícióból indulva kicsivel több, mint öt egész fordulat (-20010 inkrementig) megtételére készteti a motort óramutató járásával megegyező irányban (emiatt a negatív előjel). Látható, hogy a vezérlő utasítások rövidek, tömörek, a nyelvezet hasonlít az Assembly-re. A motort SMI-ben[18] ezzel a módszerrel is lehet vezérelni: minden kód után Enter-t ütve az adott parancs végrehajtódik. Szerencsére a motor belső logikája úgy is képes fogadni az utasításokat, ha azok egymás után, szóközzel elválasztva sorban érkeznek a bemenetére, például `VT = 1000 ADT = 6 MV G`. Ezt a tulajdonságot kihasználva LabView-ban az összes parancs string-be szervezhető és egyszerre kiküldhető.

A főképernyőn a felső, középső szekció hivatott a motor működtetésére. Az *Átküldendő adatok* egy kontroll és indikátor felület egyaránt, amely jelzi az épp a motornak küldött parancsokat és adatokat, de saját kódot begépelve, majd Enter-t nyomva innen akár programozással is vezérelhető a motor. A *Forgásirány* kapcsoló átbillentésével választható ki, hogy óramutató járásával megegyező, avagy ellentétes irányban szeretnénk forgatni a motort (a default a CCW). A két mutatós műszer indikátorként

[18] SMI: Smart Motor Interface

funkcionál, a *Sebesség* (fordulatszám) és *Gyorsulás* (szöggyorsulás) értékeit jelzik ki, melyeket a téglalapokban adhatunk meg. Itt Enter leütésén kívül az új érték a megadását követően egérmutatóval bárhova kattintva is betöltődik a változóba. A kód úgy lett megírva, hogy mindig csak akkor küldi ki ezt a két információt, amikor a *Sebesség* értéke megváltozott. Ez egy logikus megoldás, hiszen ha más gyorsítási profilt szeretnénk a motornak adni, akkor először úgyis a gyorsulás értékét kell megadni, majd azt, hogy mi az a fordulatszám, amit ilyen gyorsan kell elérnie. A *Sebesség* 10000 fordulat/perc-nél, a *Gyorsulás* 5 fordulat/s$^2$-nél lett maximalizálva. Ennél nagyobb számok megadása esetén a program azokat figyelmen kívül hagyja, és a legnagyobb lehetséges értékeket tölti be.

A *START Motor* nyomógomb egy előre beállított forgási profillal és az adott szenzor tárcsájához kiszámolt PID paraméterekkel pörgeti föl a motor tengelyét. A *TPO* gomb betölti a PID paramétereket, majd öt teljes fordulatot megtételére készteti a motort alacsony fordulatszám mellett. A *Servo OFF* gomb segítségével manuálisan is kikapcsolható a motor szervója, ha szükség lenne rá. A tőle balra található indikátor piros színnel és SERVO OFF felirattal jelzi, ha ez megtörtént, valamint zöld színnel és SERVO ON felirattal jelzi, ha a szervo üzemel. A *PortName* textbox egy kontroll elem, amely jelzi, hogy a PC-nek melyik kommunikációs portjára kapcsolódik éppen a motor (default COM6, amely begépeléssel változtatható).

Az SM23165D-vel történő kommunikáció lebonyolításához egy speciális beépülő modul alkalmazására van szükség. Ez két bemenetet fogad, egyik az *Átküldendő adatok* string, másik pedig a *PortName*. A megnevezett porton keresztül megcímzi a motort, kéri a kapcsolat felépítését, majd amikor sikeresen létrejött a vonal, átküldi az *Átküldendő adatok* változóban található értékeket. Ezt a subVI-t hívja meg minden esetben a program, amikor a motorral valamit közölni szeretne.

A nyomógombok, a forgásirány kapcsolója és a fizikai paraméterek változásainak érzékelésére egy úgynevezett Event Structure-t használunk. Ez a struktúra előre definiált események bekövetkezésére figyel (Value Change), és amint egy létrejött, a feltételében található műveletek végrehajtódnak. Mivel többször szeretnénk értékeket változtatni, ezért a struktúrát mindenképp ismétlődőre kell állítani, hogy egy eseményt többször is figyeljen. Ehhez az Event Structure modult egy különálló while ciklusba van ágyazva, amely addig fut, amíg a motor leállását jelző lokális változó igazzá nem válik (tehát a fő while befejezte működését és a motort le kell állítani).

Az alkalmazott Value Change események a következők:

- *Start Motor:* a gomb megnyomását követően minden szenzorhoz és tárcsájához egyedi inicializáló és indító kód kerül átküldésre, ugyanis minden tárcsa más súlyú és más radiális irányú tengelyterhelést visz a rendszerbe. Tehát minden tárcsa esetén a motornak más szabályzási módra van szüksége, mert eltérő nyomatékkal kell a terhelt tengelyt forgatnia. Default értékként 600 fordulat/perc-es sebességre gyorsít a motor 2 fordulat/s$^2$ gyorsulással.
- *Sebesség (rev/min):* a motorban szereplő változók a fizikai paraméterekkel arányos kapcsolatban állnak. Sebesség esetén a VT vezérlőkód 546-szorosa a valóságos értéknek, gyorsulás esetén 4 a szorzótényező. A sebesség értékének változtatása után első lépésben megtörténik a szorzás művelete, majd konkatenálódik az (4-gyel felszorzott) aktuális gyorsulással és a többi vezérlő utasítással.
- *Gyorsulás (ADT):* a beírt, új érték a gyorsulás lokális változójába kerül.
- *TPO:* a gomb logikai TRUE állapotba kerülését követően minden szenzorhoz és tárcsájához szintén egyedi PID paraméterek generálódnak. Majd ehhez hozzáíródik egy olyan művelet, amely öt egész fordulat megtételére készteti a motort 50 fordulat/perc-es fordulatszámon tengelyterheléstől függő gyorsítással.
- *Servo OFF:* a gomb lenyomása után az OFF vezérlőutasítás az *Átküldendő adatok* lokális változóba kerül.
- *STOP:* a programmegszakító/vészleállító gomb hatására a motorba az OEND X utasítás töltődik. Ez automatikus leállításra kényszeríti a motort, bármilyen funkciót is hajt végre éppen.

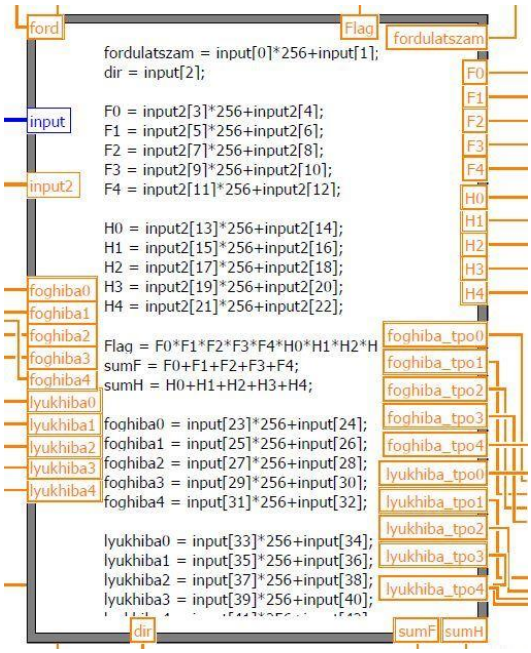
Minden eseményben a string-ek tartalma az *Átküldendő adatok*-ba kerül lokális változókon keresztül, majd meghívódik a subVI, amely soros porton keresztül lebonyolítja az adatok átvitelét. A műveletektől függően a fordulatszámhoz és szöggyorsuláshoz tartozó indikátorok értéke folyamatosan változik, és a servo állapot-visszajelzője is a motor működési fázisának megfelelően mutatja az aktuális státuszt.

#### 4.3.2.3 Analíziskörnyezet

A program túlnyomó részét az adatok fogadásáért és az analízis lebonyolításáért felelős blokkok teszik ki. A Függelékben látható két csatolt kép a 2.2.3 pontban ismertetett szenzor analízisfelületéről, a fő indítási képernyőről és a mérés befejezésekor a jelentés elkészítésére szolgáló panelról. A többi főtengely jeladónál hasonló a kijelző elrendezése és a mért adatok megjelenítése, számítása.

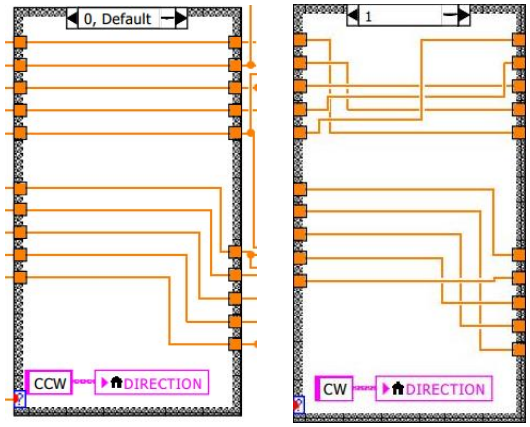
Az analízis a fő while ciklusban indul. A *START* gomb lenyomását követően a program várakozik az USART átvitelt szinkronizáló (egy új üzenet kezdetét jelző) '@' karakter megérkezésére, és amint ez az egyetlen byte beérkezett, az összes utána következőt beolvassa egy string típusú tömbbe az újabb '@' karakterig.

A fő while cikluson belül egy nagy case tartalmazza az összes szenzorra nézve a legtöbb számítási feladatot. Ez az elágazási szerkezet (és még sok másik is) a *Tab Control* értékének felhasználásával dönti el, hogy melyik kiválasztott szenzorhoz tartozó ág fog lefutni, és melyikek maradnak inaktívak. A beérkező byte-ok ugyanabban a sorrendben olvashatóak ki, mint amilyenben a portra lettek ültetve, tehát a program először szétbontja a sok byte-ból álló csomagot, majd elvégzi az átalakításokat. A mikrokontrollerben sok értéket megszoroztunk 10-zel, 100-zal, bitenkénti eltolást végeztünk rajtuk. Mindezt azért, hogy tizedes jegy pontosságban küldhessük át az adatokat, másrészről pedig, hogy minél kisebb értékeket helyezzünk a portra a kevesebb byte-szám miatt. A csomag szétbontása egy Formula Node-ban történik, amely képes C nyelvű kódokat értelmezni és feldolgozni. A 28. ábrán látható egy ilyen Node, amely a vezérműtengely jeladó által küldött adatok szétbontását végzi. Az *input* a bemeneti string típusú tömbből típuskonverzióval integer-ré konvertált tömb, az *input2* pedig az *input* tömb 10-zel osztott (tizedes jegy pontosságú) double megfelelője. A 8 bittel jobbra eltolásból az adott byte 256-tal történő megszorzásával könnyedén visszaalakítható az eredetileg kiszámított érték. Az összes többi változó, amely a Formula Node szélén helyezkedik el nem más, mint a C nyelvű kódban az egyenlőségjel bal oldalán található változónevek. Ezek mind kimeneti értékek, amelyek nagy része a mikrokontrollerben lett kiszámítva. A LabView programban minden szenzornál a Formula Node-ok a fő csomópontok, amelyek szétosztják az MCU-ból érkező összes változót a további műveletekhez.




**28. ábra: LabView-ba érkezett adatcsomag szétbontása**

A szögelfordulás értékek vezérműtengely jeladónál forgásiránytól függően más sorrendben képződnek a kontrollerben, és ennek köszönhetően más az USART továbbítás során keletkező sorrend is. Emiatt a szögelfordulások LabView-ban történő kijelzése is eltérő lesz, viszont a felhasználói felületen az adott fogakhoz és lyukakhoz tartozó indikátor dobozok, amelyek mutatják az aktuális fokokat, meghatározott helyen vannak, ezért mindig a hozzájuk tartozó értéknek kell bennük megjelennie. Ehhez egy egyszerű huzalozási módszert választottam, amellyel az adat utak pontos figyelembevételével elérhető az imént említett funkció. A 29. ábrán látható a módszer lényege, aminek eredményeként az indikátorok forgásiránytól függetlenül a nekik megfelelő szögelfordulás értékeket fogják tartalmazni.




**29. ábra: LabView, huzalozási módszerek**

A kijelző felületen minden szögelfordulás mellett található egy jelzőbit, valamint két másik téglalap. A jelzőbit zöld színű, ha az adott szögelfordulás a tűréshatárokon belül található, és piros színű egyébként. A két téglalap közül az első a fordulatonként hibásan érzékelt fog illetve lyuk hibaszámlálója, a második pedig a TPO hibaszámláló. Ezeket a LabView folyamatosan figyeli és összegezi, és ha összegük elér egy kritikus szintet, amelyet a *Hibaszámláló* határoz meg, akkor a program hibával kilép és leállítja a motort.

Szintén a 2.2.2 pontban részletezett vezérműtengely jeladóra jellemző mérési feladat a TPO módban jelentkező hibák felderítése. Sok esetben úgy érkezik be a laborba a hibás szenzor, hogy dinamikus tartományban tökéletesen működik, ám TPO esetén, tipikusan felmelegített (80…100°C) állapotban a mért szögértékek kívül esnek a tűréshatárokon. Az analíziseszközzel szemben támasztott követelményeknek eleget téve ennek a hibafajtának a felderítésére és az eredmények közlésére is fel kellett készíteni a rendszert. A motor a *TPO* gomb lenyomását követően öt egész fordulatot tesz meg, mialatt a datalogger automatikusan rögzíti a forgás befejeztéig keletkező szenzorjeleket. Az automatizálás úgy valósul meg, hogy a motor subVI meghívásakor a *LOG* nyomógomb lokális bool változója TRUE állapotba billen, majd a szekvencia vár hat másodpercet, ezalatt az öt forgás lezajlik, és végül a lokális változó visszaáll FALSE értékre. A *LOG* igazzá válása egy végtelenített while ciklusba ágyazott szekvencia kezdőfeltétele, mely után egy beépülő modul, a DAQ Assistant hívódik meg. A DAQ Assistant két bemeneti paramétert vár, a mintavételi frekvenciát és a vett minták számát. A modul kapcsolatot teremt a datalogger-rel és a két bemeneti változó értékének megfelelő mintavételezéssel a logger-en lévő jelalakokat a LabView egy kijelző felületére tölti. Minden szenzornak saját kijelzője van, ami a *Tab Control*-os feltételhez kötött case elágazásból választódik ki, ezért a *LOG* lenyomását követően mindig csak egy gráf válik aktívvá. A szekvencia egy logikai TRUE változó megadásával zárul, amely a while ciklus ismétlődési feltételéhez van kötve. Ez azt jelenti, hogy a program futása alatt bármikor lehetőség van a datalogger használatára és a jelalakok rögzítésére.
A TPO mód vizsgálatához tehát adott a szenzor jeleinek grafikus megjelenítése, de ezen kívül számértékekre is szükség van a pontos analízishez és a következtetések levonásához. Ehhez a program az első három forgás alatt kiszámított szögelfordulásokat menti el egy tömbbe. A tömb neve *TPO array* és a vezérműtengely jeladóhoz tartozó felületen található (lásd 27. ábrán). Azért az első hármat regisztrálja, mert az első hét

felfutó impulzusra mindenképp szükség van (a TPO definíciójából következik), és így szemléletes megoldást kapunk arra, hogy miként változnak meg az értékek, amikor a harmadik fordulat során már beállt a szenzor a dinamikus üzembe. Ehhez a módhoz külön riportfájl is generálódik, amely a TPO tömböt fogja tartalmazni.

A multipolrad jellegű méréseknél a grafikus felület annyiban módosul, hogy az 58 impulzus miatt sokkal több adat kijelzésére van szükség, de az eddig ismertetett programozási és feldolgozási módszerek lényegesen nem térnek el. A kijelzőn 58 szögelfordulás és mellettük kétoszlopnyi jelzőbit található. Az első oszlop piros, ha az adott szög kisebb, mint 5,7°, a második piros, ha nagyobb, mint 6,2°. Ellenkező esetekben zöld szín jelzi a helyes értékeket. További új indikátorok (melyek a vezérműtengely szenzornál nem voltak) az impulzus időtartam aktuális, minimális és maximális értékei, a *Hibaszámláló_több*[19], *Hibaszámláló_kevesebb*[20], a *Szinkronhiba*[21], a *p_abszolút* és a *p_ismétlési*[22]. A *HIBAKORLÁT*nak a hibaszámlálók és a szinkronhiba összegénél kell kisebbnek lennie ahhoz, hogy a program befejezze a működését és leállítsa a motort.

A vizsgálatok során többféle jelentés is készül, illetve készíthető:

- *.txt formátumú Log fájl*:

  A mérés során két másodpercenként regisztrálja a LabView felületen megjelenő összes olyan értéket, amelyből hibás működésre lehet következtetni. Főtengely jeladónál a hibaszámlálók értékeit, vezérműtengely jeladónál pedig az egyes fogak és lyukak szögelfordulásait, valamint mindkét esetben az aktuális fordulatszámot és a megtett fordulatok számát. Ez az adatrögzítés akkor hasznos, ha hosszabb ideig, akár napokig szeretnék használni egy szenzor vizsgálatára a rendszert, mert bármilyen hiba érkezik az analízis során, arról csak az adott pillanatban kapunk értesítést, viszont ezzel a módszerrel a teljes mérési folyamat visszakövethető a kezdetekig. Azért csak két másodpercenként történik mentés, mert gyorsabb archiválás esetén túl nagy méretű fájlt kaphatnánk. A Log fájl egy

[19] Egy fordulat alatt 58-nál több impulzus érkezése esetén eggyel növeli tartalmát.

[20] Egy fordulat alatt 58-nál kevesebb impulzus érkezése esetén eggyel növeli tartalmát.

[21] Eggyel növeli tartalmát, ha a szinkronimpulzus helyzete az első fordulat alatt mérttől ±10 enkóder inkrementtel eltér.

[22] Abszolút és ismétlési pontosság

sora alább látható (vezérműtengely mérése esetén). Az értékek: fordulatszám, megtett fordulatok száma, fogak és lyukak szögelfordulásai.


- *.html vagy .pdf formátumú, protokoll szerinti Riport fájl*:

  Olyan formális dokumentáció, amely minden mérés befejezését követően rögtön elkészíthető. Ez az űrlap szolgál referenciaként a gyártók, beszállítók tájékoztatásakor és az esetleges reklamációkkal szemben is. A riport generálásához az analízisfelületen egy saját oldal áll rendelkezésre (lásd Függelék), ahol alap információk megadását követően egy gombnyomásra elkészül a jelentési ív. A *STOP* gomb lenyomása után, vagy ha a *HIBAKORLÁT* értékét túllépik a keletkező hibák, a program egyből erre a lapra ugrik. Ezzel tájékoztatja a felhasználót, hogy befejeződött a mérés, valamint nyomatékosít, hogy minden esetben készüljön jelentés és legyen nyoma a vizsgálatnak. A Riport felületen meg kell adni szöveges formátumban az alkatrész hatjegyű regisztrációs számát (QTS szám), a dátumot, az analízist végző nevét, a hőmérsékletet, amelyen a mérés folyt, a reklamáció tárgyát, mellyel az alkatrész a laboratóriumba érkezett és a vizsgálat eredményét. Az adatok kitöltése nem kötelező (de a pontos beazonosítás végett mindenképp célszerű), akár mindent üresen hagyva is elkészíthető a jelentés.

  A riport generálása egy teljesen különálló szekvencia, amely a fő while-ból való kilépést követően kapja meg a futási jogot. Az egész program működése során csak egyszer hajtódik végre, mivel egy mérésről egyetlen riportnak kell készülnie. A szekvencia első lépésben kiüríti a felületre korábban beírt adatokat, majd jpeg fájlba menti a datalogger-es hullámformát. A *Tab Control* lokális változójának tartalmától függően (tehát a vizsgálható szenzorok fajtái miatt) ötféle dokumentáció készíthető, valamint ehhez még hozzájön a TPO mód esetén menthető is. Minden szenzornál tömbök tartalmazzák a szögelfordulások értékeit, melyek a sorok és oszlopok címeivel kiegészülve string-gé konvertálódnak, majd a *Mérési eredmények* tömbbe kerülnek. Ez a string egy táblázatként jelenik meg mind a LabView *REPORT* felületén, mind a legyártott fájlban. A táblázat három oszlopból áll (kivéve TPO esetén, ahol a három fordulat miatt négy oszlop lesz), az első a nominális, a második a mért szögelfordulásokat tartalmazza, a harmadik oszlop pedig a helyességet OK, a

hibát ERR felirattal. Multipolrad esetén az első oszlop az impulzus sorszámát és a tömbben való elhelyezkedését tartalmazza (0-57), a második a mért szöget, a harmadik pedig szinkron esetén SYNC, a többi esetben pedig szintén OK, illetve ERR értéket. Ha a *Mérési eredmények* tömb elkészült, akkor ezt követően a program a *Report elkészítése* gomb lenyomására fog várakozni. A gomb felett található választókapcsoló állásától függően a generált fájl html, vagy pdf formátumú lesz. Mindkét esetben a program az A4-es lapon koordináták megadásával helyezi el a szöveges információkat és a képeket. Minden adatot és a köztük lévő tabulátort, sortörést egymás után fűzi, majd egy új tömbként küldi ki a megadott koordinátára.

A fájl a generálása előtt elérési útvonalat és nevet kap. A program fő könyvtárán belül a Reports mappába kerül, ahol a következő formátumú nevet kapja a könnyű beazonosíthatóság miatt:

<QTS szám>_<dátum ÉÉÉÉ.HH.NN.ÓÓ.PP.MM formában>. Például 123456_2013.11.23.08.35.40.pdf, majd ezután elkészíti a jelentést a kívánt formátumban.

A riport fájl egy fejléccel kezdődik a vizsgálatot végző laboratórium megjelölésével, a mérés jellegének megnevezésével és egy AUDI-s logóval. Ezt követik a felhasználó által begépelt adatok, a mérőberendezés leírása, a mérési időtartam és az adott analízishez tartozó számított értékek, melyek a LabView felületen is kijelzésre kerültek (fordulatszám, a *Mérési eredmények* táblázat a szögelfordulások értékeivel…). Az adatok után pedig a datalogger-rel mintavételezett hullámforma található (lásd Függelékben).

- *.jpg formátumú képfájlok a datalogger-es hullámformákról*:
  Ugyan az összes riport képként beágyazva tartalmazza a rögzített jelalakokat, a program mégis mindet elmenti jpg formátumban is egy külön mappába az előbb ismertetett elnevezési stílusban azért, hogy ne kelljen ezeket külön kimenteni az elkészült jelentési lapokról.

Mivel a program sok kimeneti fájlt készít, ezért egy megfelelő könyvtári struktúrára is szükség van a jó átláthatósághoz és kereséshez. A LabView program fő könyvtárán belül három mappa található Logs, Reports és Images néven. Értelemszerűen mindegyik a neki megfelelő adattípust tartalmazza. A Reports-on belül az összes dokumentáció saját mappát is kapott. Erre azért volt szükség, mert html-ben történő mentés esetén a

html fájlon kívül még külön elmentődnek a benne található képek is (fejléc, hullámforma). Ezzel a megoldással minden egy helyen tartható és elkerülhető az adatok elkavarodása. Az alfájlok elnevezése szintén a korábban megadott formátumú, így tökéletesen beazonosítják a vizsgált szenzort, a vizsgálat idejét és az elkészített riport fájlt is.

## 4.4 Összefoglalás

A negyedik fejezet bemutatta az elkészült rendszer mechanikai egységeit, a két program felépítését és működését, a hozzájuk kapcsolódó fontosabb változókat és számítási algoritmusokat. Az ötödik fejezet egy átfogó összegzést ad arról, hogy honnan indult a fejlesztés és hogy meddig sikerült eljutni, ezután kitér arra, hogy milyen esetleges továbbfejlesztési lehetőségeket rejt még magában a rendszer.

# 5 Mérési eredmények

Az elkészült rendszer minden követelménynek eleget tesz, amelyekről a 3.1 fejezetben szó esett. A mikrokontrolleres egység képes fogadni a vizsgált szenzor és a kvadratúra enkóder jeleit, elvégezni az egyes paraméterekhez szükséges számításokat és továbbítani az adatokat a LabView környezetnek. A PC-n futó program tudja fogadni a soros porton érkező információkat, azokkal további műveleteket végezni és a megfelelő hibajelző funkciókkal a képernyőre íratni és a felhasználót tájékoztatni az esetleges hibákról, valamint vezérelni a motort és a kontrollert egyaránt. A mérésről jelentés is készíthető, mely segítségével részletes ismertetést kapunk az analízist követően, amelyet a beszállítók reklamációival szemben és az egyes hibaképek ismertetése során fel lehet használni. A rendszer nagy fordulatszámok esetén is ugyanolyan pontossággal és gyorsasággal képes az adatok feldolgozására, a tesztelések során a LabView program legnagyobb processzor kihasználtsága egyszer sem haladta meg a 10%-ot. Ez köszönhető annak, hogy a programozás során minden egyes ismétlődést végrehajtó ciklusban msec nagyságrendű késleltetéseket alkalmazva elérhetővé vált, hogy a program folyamatosan rövid időre visszaadja a vezérlést a processzornak, így elkerülve a felesleges leterhelést.

A berendezés a tesztek alatt már többször is rendkívül megbízhatónak és hatékonynak bizonyult. Nem egy esetben sikeresen megtalálta a szenzorok által produkált hibákat, és feltárt egy olyan gyártási problémát is, aminél sok szenzor sorozatosan ugyanazzal a hibával esett ki. A beszállítót a laborba hívva a rendszernek köszönhetően alátámasztottuk a pontatlanságot és a gyártó az eszköz által szolgáltatott meggyőző eredményekre hivatkozva biztosított minket a hiba kijavításáról.

A fejlesztés még nem ért a végső fázisába, ugyanis a többfunkciós nyomtatott áramkör illesztésére és a műszerdoboz elkészítésére sajnos már nem maradt elég idő. Egyelőre egy prototípusa létezik az áramkörnek és a műszerdoboznak egyaránt. A NYÁK a szenzorok jeleinek illesztését és szűrését végzi, de a hőmérséklet érzékelését és a ventilátor vezérlését még nem sikerült megvalósítani. Ehhez szükség van az STM32F103 AD-átalakítójának megismerésére és a kapcsolódó programkódok elkészítésére, melyekkel elérhető, hogy a termisztor által szolgáltatott feszültségjelekből a hőmérséklet kiértékelését követően a mikrokontroller automatikusan ki és be tudja

kapcsolni a ventilátort. Továbbá az IRFZ24NS MOSFET használatához a NYÁK-ot újra kell tervezni, mert a 3,3 V-os ráadott feszültség hatására még sajnos nem nyit ki teljesen, ami azt eredményezi, hogy a ventilátor sem tud a legnagyobb fordulatán üzemelni és ezért nem képes maximális hűtést biztosítani. Erre már született is egy ötlet, amiben a 3,3 V-os GPIO láb egy NPN tranzisztor bekapcsolását vezérelné. A tranzisztor kollektor ága egy ellenálláson keresztül 5 V-ra húzná fel a MOSFET Gate-jét, amely már elegendő lenne a teljes nyitáshoz. Az adott GPIO láb vezérlése egyelőre a LabView felületen keresztül lehetséges a Függelék 9.1-es ábráján látható Ventilátor gombbal. Amíg a kontrollerben az AD-átalakító nem kerül felprogramozásra, addig a gomb segítségével lehet a ventilátort manuálisan ki- és bekapcsolni.
Ezen felül találni kell megfelelő műszerdoboz kialakítást is, amely átlátszó plexiborítással rendelkezik, hogy védje a jelfeldolgozó áramköröket és láthatóvá tegye a kontroller LCD kijelzőjét. Amint ezek a feladatok megvalósultak, nyugodtan kijelenthető, hogy a fejlesztés befejeződött és a rendszer elkészült.

Az analízist végző berendezéssel kapcsolatos további fejlesztési lehetőségekkel a pontosság és a mérési felbontás növelhető, a számítási algoritmusok nem igényelnek változtatást, de a felhasználói felület még a felmerülő igények szerint módosítható. Az egyik ilyen irányú alternatíváról röviden már említést tettem a 4.1.1 részben, ahol a mechanikai egységet mutattam be. A metszeti kép (21. ábra) egy olyan alkatrészt is jelez a tengelyen, amely még nem került beépítésre. Ez egy 300 000-es felbontású inkrementális enkóder, melynek megrendelése már megtörtént, viszont a szakdolgozat befejezéséig sajnos még nem érkezett meg. Célja a pontosság oly szintű növelése, mellyel képessé válik a rendszer arra, hogy meghaladja a szenzorgyártó tesztelésre használt eszközének a pontosságát. Másik lehetőségként a rendszer automatizáltsági fokát lehetne növelni oly módon, hogy a tipikus hibaképeket önállóan felismerje, és a hozzá kapcsolódó információkkal a riportot magától kitöltse. Ezzel a módszerrel a felhasználónak tényleg csak pár gombot kellene lenyomnia, és az eszköz a számítási és így a kiértékelési feladatokat is külső beavatkozás nélkül végezné el.

Összességében elmondható, hogy az elkészült berendezés funkcionálisan üzemképes, a szükséges műveleteket és kiértékeléseket a követelményeknek megfelelően képes elvégezni, viszont még számos lehetőséget rejt magában, melyek segítségével a fordulatszám jeladók analízise még pontosabbá és megbízhatóbbá válhat.

# 6 Összegzés

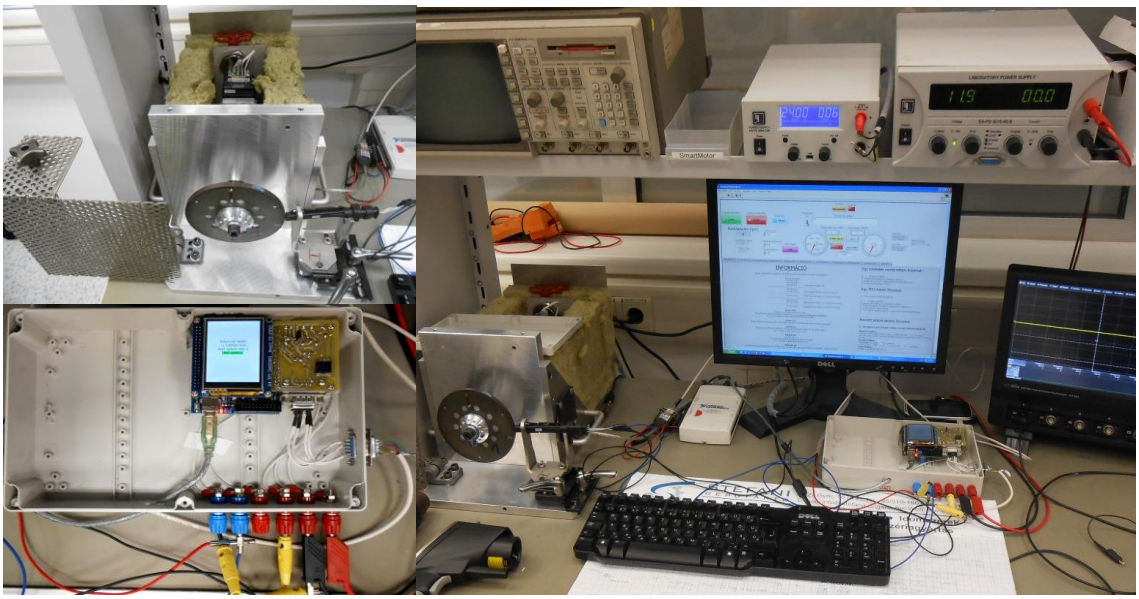
A teljes berendezés kifejezetten az Audi Hungária Motor Kft. elektromos laboratóriumában dolgozó mérnökeinek és technikusainak készült, és a cél az volt, hogy a mindennapos fordulatszámjeladó-mérések a pontosság és megbízhatóság növelése mellett minél egyszerűbben és minél kevesebb lépésben elvégezhetőek legyenek.

A kiindulási állapotban egy egyszerű és teljesen analóg konstrukcióval lehetett az analízist elvégezni. A feladatom az volt, hogy egy minden tekintetben új, digitális alapokra ültetett rendszert fejlesszek ki, mellyel a szenzorok legfontosabb tulajdonságai (szögelfordulás, impulzusszélesség, abszolút és ismétlési pontosság) számszerű értékekkel meghatározhatóvá válhatnak, amire a korábbi rendszer nem volt képes. A fejlesztés egy új rendszerterv kidolgozásával indult, melyben helyett kapott egy beágyazott mikrokontrolleres panel, egy jelillesztést végző nyomtatott áramkör, és egy egyedileg fejlesztett és kérésre legyártatott mechanikai egység, kvadratúra enkóderes, programozható szervomotorral. Ezt követte a megvalósítás fázisa, amely túlnyomórészt a beágyazott programkód és a LabView-ra fejlesztett kód megírását jelentette. A rendszer élesztése és analízisbe való bekapcsolása sorozatos teszteléseket követően megtörtént, a motor és a programok az elvártnak megfelelően, összehangoltan végzik a dolgukat. További megoldandó feladat a nyomtatott áramkör teljes értékű funkcionális elemmé alakítása, hogy a termisztorral és a ventilátor vezérlésével a gépház belsejében uralkodó hőmérséklet szabályozhatóvá válhasson, valamint a műszerdoboz elkészítése, hogy a bonyolult vezetékezés megszűnjön, és a beágyazott és illesztő panel egy egységen belül, védett helyen legyen. A jövőben a pontosság növelése érdekében egy új, sokkal nagyobb felbontású enkóder kerülhet beépítésre, valamint a kódok optimalizálásával és továbbfejlesztésével a rendszer nagyobb szintű automatizáltsága érhető el.

A közel egy évig tartó fejlesztés során rendkívül sok új mérnöki megoldással gazdagodott a tudásom. Ízelítőt kaptam a komplex rendszerek megtervezéséhez szükséges gondolkodásmódból, hogy milyen lépéseket kell megtenni ahhoz, hogy a szinte nulláról történő építkezés végül egy összetett, sok modulból álló architektúra összehangolt működését eredményezze. Továbbá megismerkedtem egy számomra eddig teljesen ismeretlen 32 bites mikrokontroller belső felépítésével, a hozzá kapcsolódó fejlesztőkörnyezettel és az STM családra jellemző beágyazott C programozási nyelvvel.

A beágyazott programkódok fejlesztése során sok új módszerrel és lehetőséggel találkoztam. Talán a legfontosabbak közülük a megszakításkezelésen alapuló eseményvezérelt programozás, a timer egységekhez kapcsolódó inicializálási és vezérlő műveletek, a soros porton történő kommunikáció felépítése és az adatok átvitele, az LCD kijelző meghajtása, a kijelzőre történő kiíratás megvalósítása, és nem utolsó sorban a strukturált, menürendszeren alapuló, sok megszakítási vonalat kezelő programszerkezet létrehozása. A felhasználói felület elkészítéséhez és az adatok rendezett formában történő megjelenítéséhez az általam még sosem használt LabView fejlesztőkörnyezet kínálta lehetőségeket is meg kellett ismernem. A grafikus programozás szakít az eddig tanult kódolási technikákkal, viszont nagyon hasznos, és a későbbiekben is alkalmazható tudással látott el. Elsajátítottam az alapfunkciókon túl a datalogger-es mintavételezéshez, a VISA soros port kommunikációhoz szükséges ismereteket, továbbá sikeresen implementáltam a mikrokontroller és a SmartMotor vezérlését megvalósító kódrészleteket, és a riport generálásához – a LabView-ban egyelőre nem beépített funkcióként működő – a pdf-ben történő mentés műveletét is.
A programozási feladatokon túl elektronikai tudásomat is bővítettem a NYÁK megtervezése és legyártása során.

A fentiek alapján összességében elmondhatom, hogy a szakdolgozatom témája egy nagyon komplex, szerteágazó feladat megvalósítása volt, melynek köszönhetően rendkívül sok szakmai tapasztalattal gazdagodtam, és reményeim szerint ezen információkat későbbi pályafutásom során is sikeresen alkalmazhatom majd.



**30. ábra: Az elkészült mérőeszköz**

# 7 Ábrajegyzék

# 8 Irodalomjegyzék


[1] Szauter Ferenc, *Kerékfordulatszám szenzorok,* Széchenyi István Egyetem Közúti és Vasúti Járművek Tanszék, 2011, [Online – PDF]
http://rs1.sze.hu/~szauter/tetelek/041_TETEL.pdf
[Elérés ideje: 2013. augusztus 5.]

[2] Dr. Varga Zoltán, Szauter Ferenc, *Járműmechatronika,* Széchenyi István Egyetem, [Könyv, Online]
http://www.tankonyvtar.hu/hu/tartalom/tamop425/0007_09-Jarmumechatronika/adatok.html
[Elérés ideje: 2013. augusztus 17.]

[3] *Hall-szenzor fogalma és Hall-IC-k működése,* Fizipedia BME, 2013, [Online]
http://fizipedia.bme.hu/index.php/Hall-effektus
[Elérés ideje: 2013. szeptember 18.]

[4] Szintén Gábor, *Hall-szenzor felhasználása az autóiparban,* Szent István Egyetem Fizika és Folyamatirányítási Tanszék, 2011, [Online – PPT].
http://fft.szie.hu/fizika/meroerz/06_07/Hall%20szenzor%20bemutat%f3.ppt
[Elérés ideje: 2013. augusztus 7.]

[5] *Animatics SmartMotor SM23165D,* Moog Animatics Incorporated, 2013, [Online].
http://www.animatics.com/products/smartmotor/animatics/nema-23-2300-series/sm23165d.html
[Elérés ideje: 2013. október 1.]

[6] *Siemens SIMOTICS S 1FK7 Servomotor,* Siemens AG, 2013, [Online].
http://www.automation.siemens.com/mcms/mc/en/motors/motion-control-motors/simotics-s-servomotors/simotics-s-1fk7/pages/simotics-s-1fk7.aspx
[Elérés ideje: 2013. október 1.]

[7] *PLC: Programmable Logic Controller,* Wikipedia, 2013, [Online].
http://hu.wikipedia.org/wiki/Programmable_logic_controller
[Elérés ideje: 2013. október 1.]

[8] dr. Csubák Tibor, *Programozható irányítóberendezések és szenzorrendszerek,* BME-VIK IIT, 2013, [Online – PDF].
http://sirkan.iit.bme.hu/dokeos/courses/BMEVIIIA349/document/Programozhat%F3_ir%E1ny%EDt%F3berendez%E9sek_-_eg%E9sz.pdf?cidReq=BMEVIIIA349
[Elérés ideje: 2013. október 1.]

[9] Dr. Lantos Béla, *Irányítási rendszerek elmélete és tervezése I.,* Akadémiai Kiadó, Budapest, 2009, [Könyv].

[10] *Kvadrature encoder,* Cytron Technologies, 2012, [Online].
http://tutorial.cytron.com.my/2012/01/17/quadrature-encoder/
[Elérés ideje: 2013. október 10.]

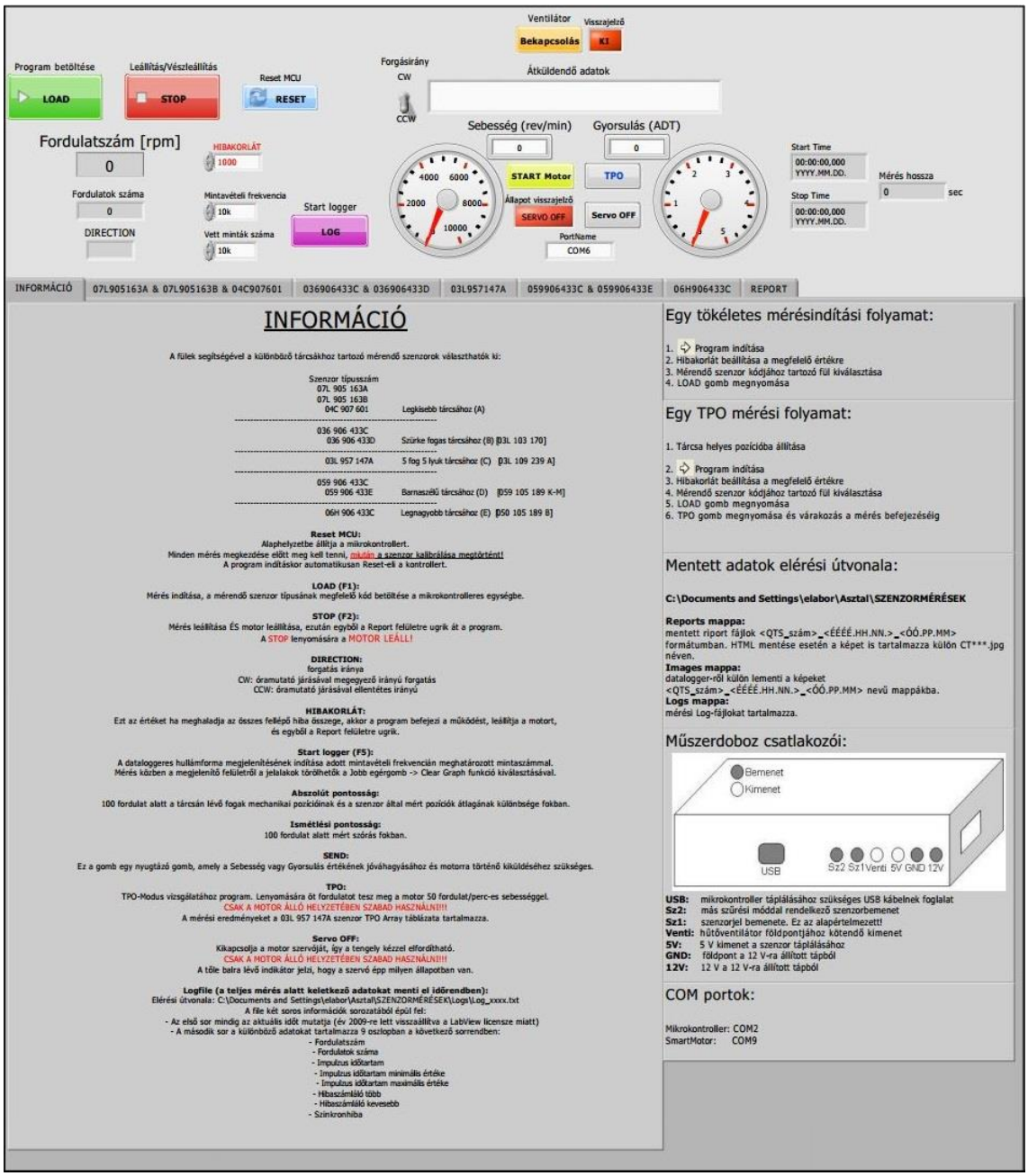
[11] *Inkrementális forgó jeladók,* Q-Tech Automatika, 2013, [Online – PDF]
http://www.q-tech.hu/pdf/PR/Inkrementalis%20forgojeladok.pdf
[Elérés ideje: 2013. október 10.]

[12] *STM32F103xx Reference Manual,* ST Microelectronics, 2011, [Online – PDF]
http://www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf
[Elérés ideje: 2013. november 2.]

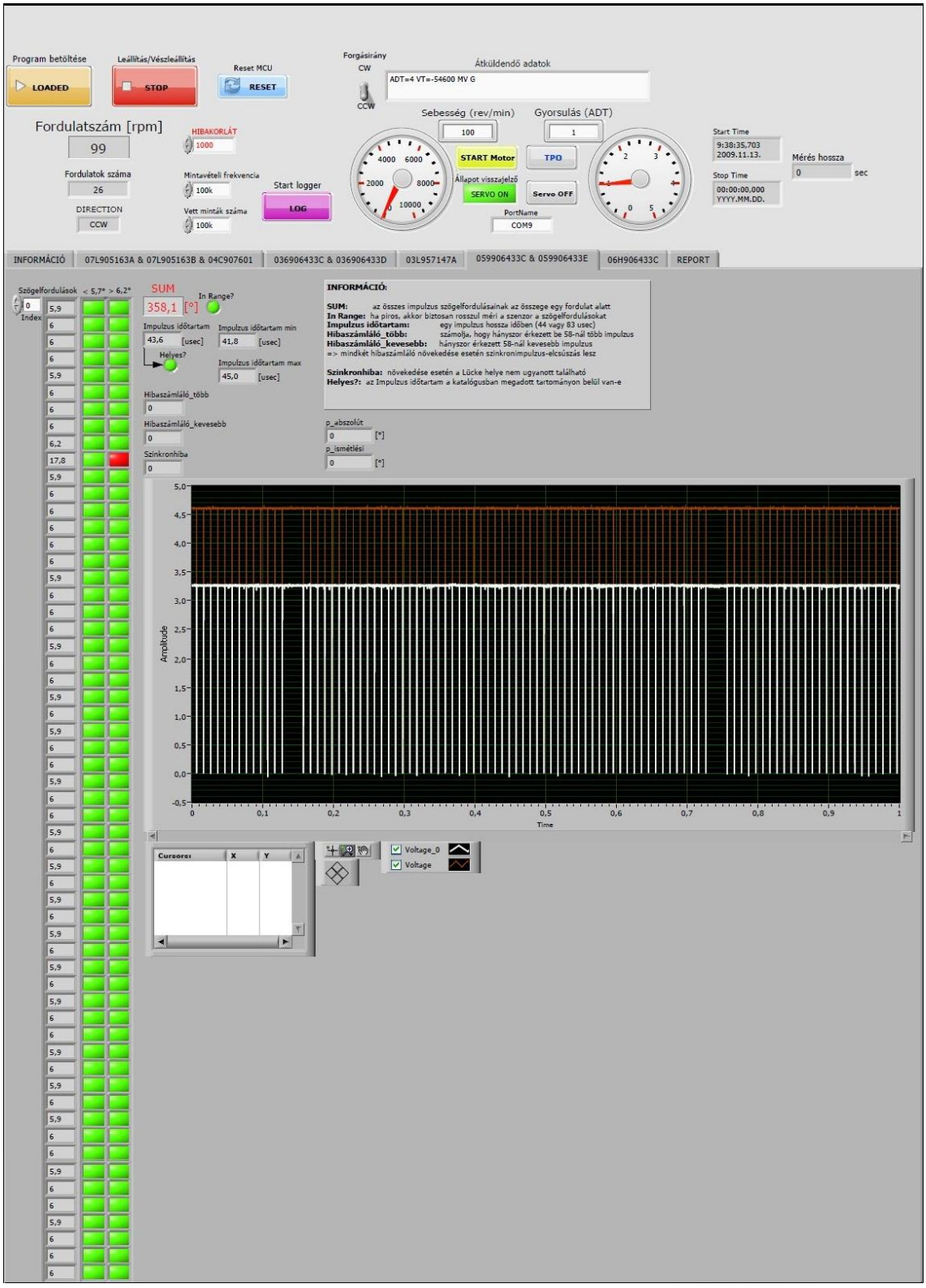
[13] *LabView,* National Instruments, 2013, [Online]
http://hungary.ni.com/labview/letoltes
[Elérés ideje: 2013. november 11.]

# 9 Függelék

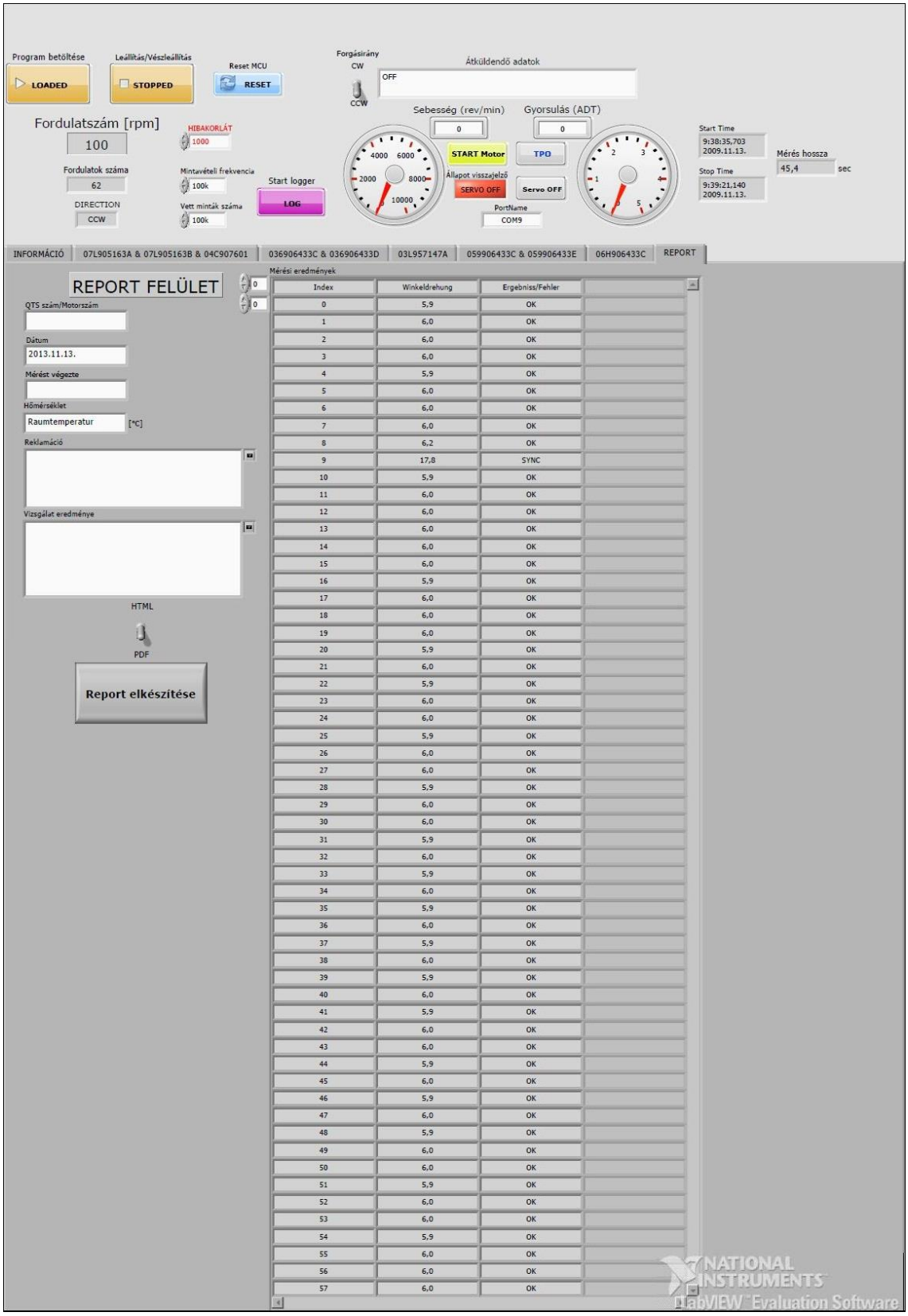
## 9.1 LabView felület fő indítási képernyője



## 9.2 LabView felület a 2.2.3 pontban ismertetett főtengely jeladó vizsgálatához



# 9.3 LabView felület a riport elkészítéséhez



# 9.4 Generált Riport fájl

| Qualitätssicherung<br>G/GQ-P33<br>E-Labor | Elektrik<br>Analysebericht | Audi |
|---|---|---|

QTS/Motornummer: 4867766
Datum: 2013.10.18.
Messungersteller:
Temperatur: Raumtemperatur °C
Messgerät: SmartMotor SM23165D, STM32F103xx MCU, NI USB-6212 Datenloggermodul
Messzeit: 1221,422 sec

Beanstandungstext:

Analyse/Ursache:
Warmen zustand Segmentlaenge n.i.O.

Messergebnisse:

Drehzahl: 499 rpm

Sum Zähne: 203,0°
Sum Lücke: 591,7°
SUM: 794,7°

| Zähne | Gemesstwerte | Ergebniss/Fehler |
|---|---|---|
| 18 | 17,4 | ERR |
| 45 | 134,7 | ERR |
| 6 | 10,0 | ERR |
| 24 | 26,7 | OK |
| 12 | 14,2 | OK |
| Lücke | ---------- | ---------- |
| 72 | 72,8 | ERR |
| 27 | 297,7 | ERR |
| 12 | 80,1 | ERR |
| 66 | 65,5 | ERR |
| 78 | 75,6 | OK |