



SZÉCHENYI ISTVÁN EGYETEM
MŰSZAKI TUDOMÁNYI KAR
Automatizálási tanszék



OTDK-Dolgozat

Egyenáramú motor fordulatszám szabályozás

Szüts Gábor

**Villamosmérnöki BSc szak
Automatizálási szakirány
4. évfolyam**

Konzulens: Marcsa Dániel, egyetemi tanársegéd

2014, Győr

Bevezető

Egyenáramú motor fordulatszám szabályozás

Dolgozatom témája egy olyan eszköz megtervezése és elkészítése ami a jövőben felhasználható az egyetemi szabályozástechnika oktatásban demonstrációs célokra. Az általam tervezett eszköz egy DC motor fordulatszám szabályozó. A tervezéskor figyelembe vett szempontok közé tartozott, hogy az eszközzel bemutathatók legyenek a P, PI, PID szabályozók, valamint, hogy a hallgatók akár saját algoritmusokat is kipróbálhassanak rajta. Szintén szempont volt, hogy az eszköz a későbbiekben továbbfejleszthető legyen, alkalmas legyen újabb funkciók megvalósítására. A feladat megvalósítására a Microchip PIC16F877 mikrovezérlőjét használtam fel. Ez a processzor 40db lábbal rendelkezik, amik közül szinte mind felhasználható digitális ki/bemenetként. A vezérlőbe került még 10 bites ADC modul, 2db CCP modul, valamint USART modul is. Ezzel az eszközzel tehát az alap funkciókon kívül számos további funkció is megvalósítható. A fordulatszám alapjelet potenciométerrel lehet beállítani, ezt az analóg jelet a mikrokontroller AD átalakítója dolgozza fel. A motor tengelyére egy tárcsát szereltem fel amelynek szélére egyforma távolságokra lyukakat fúrtam. Ennek a perforált tárcsának és egy optokapunak a segítségével mérem a motor aktuális fordulatszámát. A DC motort PWM jellel vezérelem, így lehetőségessé válik a motor fordulatszámának mikrovezérlő szoftverből való beállítása. A PWM jelet a PIC16F877 Capture/Compare/PWM modulja állítja elő, majd ez a jel egy FET-es kapcsoláson keresztül vezéri a motort. Mivel a mikrovezérlő legtöbb lába szabadon maradt volna ezért a hardvert elláttam egy karakteres LCD kijelzővel, nyomógombokkal, állapotjelző LED-ekkel, és egy RS232 soros porti kommunikációs modullal. Ezek számos fejlesztési lehetőséget biztosítanak a jövőre nézve, pl: HMI kezelő felület kialakítása a kijelzővel és a nyomógombokkal, PC-s szoftverrel való kommunikáció soros porton keresztül, mért adatok feldolgozása és megjelenítése PC-n, stb... A hardver kapcsolási rajzát, és a NYÁK tervet az EAGLE Layout tervezőszoftverrel készítettem. A mikrokontrolleren futó szoftvert a Microchip MPLAB X IDE fejlesztő környezetében írtam C nyelven, fordításra a Microchip XC8 fordítóját használtam.

Tartalomjegyzék

1. Bevezetés	1
2. A hardver tervezése	1
2.1. A motor.....	2
2.2. Motor paraméterek mérése	3
2.2.1. A forórésztekercselés ellenállásának mérése.....	3
2.2.2. A forórésztekercselés induktivitásának mérése.....	4
2.2.3. A motorállandók mérése.....	6
2.3. Motor hajtás elektronika.....	7
2.4. Tápegység.....	9
2.5. Vezérlő áramkör	10
2.5.1. PIC 16F877.....	11
2.6. Fordulatszám mérő modul.....	14
2.7. Alapjel képző modul.....	15
2.8. Soros kommunikációs modul	16
2.9. HMI modul	19
2.9.1. Karakteres LCD kijelző	20
2.9.2. Beviteli eszköz.....	25
2.9.3. LED modul	27
3. Nyomtatott áramkör tervezése	28
3.1. Sprint layout szoftver bemutatása	28
3.2. A tápegység NYÁK terve.....	32
3.3. A motor hajtás panel NYÁK terve	33
3.4. A vezérlő panel NYÁK terve	33
3.5. A HMI panel NYÁK terve	34
4. A mikrokontroller szoftvere	36
4.1. Az MPLAB X fejlesztői környezet bemutatása	36
4.2. A szoftver felépítése	40
4.3. Az alapjel feldolgozó AD modul.....	42
4.4. A fordulatszám mérés.....	47

4.5. A szabályozás	53
4.6. A PID szabályozó	55
4.6.1. P szabályozó tag	56
4.6.2. Az I szabályozó tag.....	57
4.6.3. A D szabályozó tag.....	58
4.7. PID szabályozó szoftveres implementálása.....	59
4.8. A PWM jel előállítása	60
4.8.1. PWM jel előállítása a CCP modul használatával	61
4.8.2. A PWM modul konfigurálása.....	64
4.9. LCD vezérlő programrészlet	65
4.10. Az USART modul programozása.....	68
5. Összefoglalás	71
Irodalomjegyzék	72
Mellékletek	73
I. Melléklet, a hardver teljes kapcsolási rajza	73

1. Bevezetés

A projekt amit ez a dolgozat bemutat egy tanszéki demonstrációs eszköz tervezése volt. Egy olyan eszköz elkészítése volt a cél amelyet fel lehet használni a szabályozástechnika oktatásban. A diákok ezen az eszközön kipróbálhatják a különböző szabályozó algoritmusokat, és megtanulhaták azt is, hogyan programozzanak akár saját algoritmusokat is. Az általam tervezett eszköz segítségével sokkal gyakorlatközpontúbbá válhat a szabályozástechnika oktatás a jövőben. A dolgozat első nagy fejezete a hardver tervezéséről fog szólni. A hardver tervezése az egész projekt első nagy lépése volt. Meghatároztam, a projekt célját, az elvárásokat az eszközzel szemben, majd ezeket szem előtt tartva kezdtem el a kapcsolásokat megtervezni, méretezni, és kiválasztani a felhasználandó alkatrészeket. A következő lépés ami még a hardver tervezéshez kötődött a NYÁK panelek megtervezése volt. A hardver tervezése után kezdtem el a szoftvert megírni. Itt implementáltam a projekt elején meghatározott funkciókat. A dolgozatban ezeken funkciók megvalósítását külön fejezetekben taglalom. A szoftverről szóló fejezetben térek ki a szabályozás elméleti hátterére és itt mutatom be a különböző szabályozó típusokat is, mivel a szabályozás algoritmusát is szoftveresen implementáltam. A dolgozat utolsó fejezetében a projekt során szerzett tapasztalatokat írom le, valamint a jövőbeli fejlesztési tervekről, lehetőségekről értekezek.

2. A hardver tervezése

A következő fejezetekben a hardver tervezéséről fogok részletesen írni. A teljes hardvert kisebb blokkokra bontva fogom bemutatni. Kitérek majd a kapcsolási rajzokra, bemutatam a felhasznált alkatrészeket, és részletesen ismertetem a tervezési szempontokat. Először a motorról és a motor hajtás elektronikáról írok, majd a tápegységről, később pedig a vezérlőről, és a hozzá kapcsolódó perifériákról.

2.1. A motor



1. ábra. A felhasznált motor

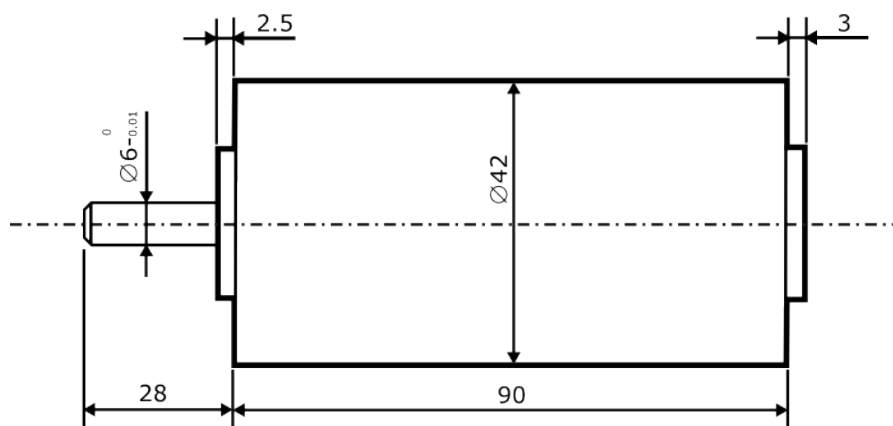
A motor kiválasztásánál az elsődleges szempont az alacsony feszültség volt, mert az ilyen feszültségi szintre viszonylag könnyen lehet vezérlő elektronikát és tápegységet készíteni, valamint az alkatrészek ára is viszonylag alacsony. Ezért az eszközhöz egy 12V feszültségű, Transtecno EC020.120 típusú állandó mágnesű DC motort választottam. A katalógus adatokat az *1. táblázat* tartalmazza.

1. táblázat. EC020.120 motor katalógus adatok

P_n [W]	20
U [V]	12
I [A]	2.6
M_n [Nm]	0.06
N [fordulat/perc]	2850

Ha végignézzük az *1. táblázat* adatait, láthatjuk, hogy egy viszonylag kis teljesítményű motorról van szó, amelynek a nyomatéka is meglehetősen alacsony. Mivel a motort nem terheljük üzemszerűen, a demonstrációs célokra ekkora nyomaték is megfelelő lesz. A motor fordulatszámát 700 és 2748 fordulat/perc értékek között képes szabályozni a kontrolller.

A motor méretei az *2. ábrán* láthatóak. A tengely 6 mm átmérőjű, erre gyártattam egy tengelykapcsolót, amivel fordulatszám méréshez használt tárcsát rögzítem a motor tengelyéhez.



2. ábra. A motor méretei

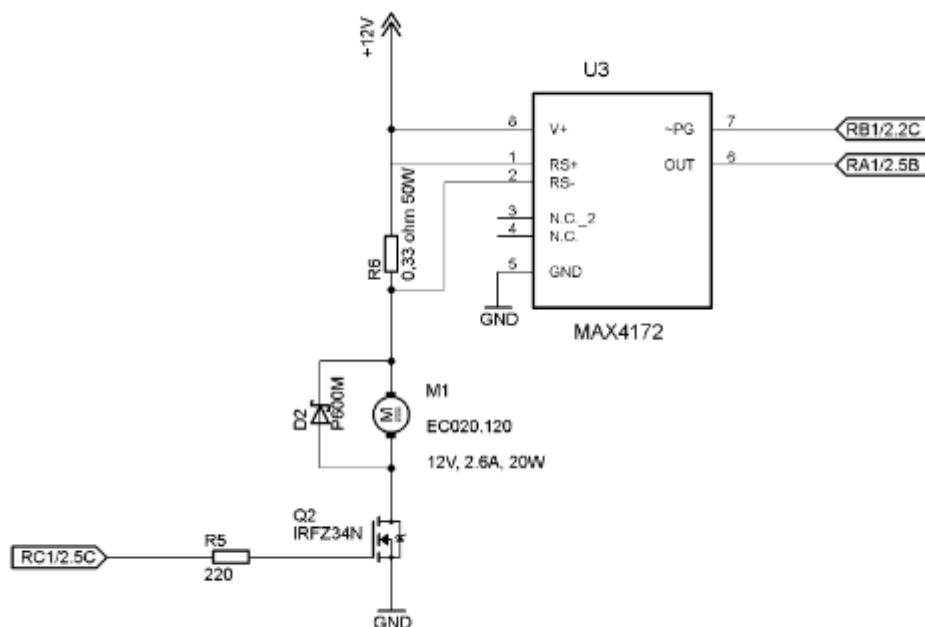
5. táblázat. Motorállandók, számított értékek

Feszültség [V]	$K = K_e = K_t$ Motorállandó értéke
2	0,014767
4	0,023077
6	0,025321
8	0,026475
10	0,027153
12	0,027512

A számított értékekből tisztán látszik, hogy a K_e érték valóban egy feszültségtől független állandó értékű motor paraméter. Ennek értéke: 0,024051 V/(rad*s).

2.2. Motor hajtás elektronika

A motor hajtás elektronika kapcsolási rajza a 3. ábrán látható. A kapcsolási rajzokat az EAGLE Layout szoftverrel készítettem, amelynek freeware verziója letölthető a gyártó weboldaláról. A teljes kapcsolási rajzot, nagy terjedelme miatt kisebb részekre osztottam, és az oldalak közötti kapcsolódásokat kereszthivatkozásokkal jelöltem.



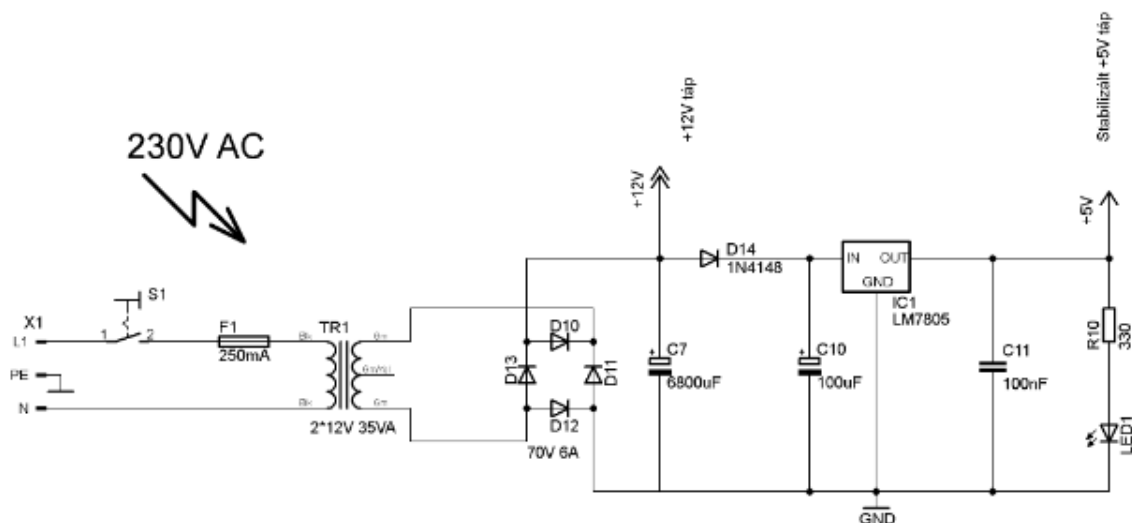
3. ábra. Motor hajtás elektronika, kapcsolási rajza

A feladathoz az egyik lehető legegyszerűbb motorvezérlő kapcsolást alkalmaztam, mert ez az egyszerű elektronika is képes ellátni a feladatot. Mivel forgásirány váltást nem akartam megvalósítani, ezért nem volt szükségszerű H-hidas kapcsolást használni. A kapcsolást forrása: [5]. A kapcsolatban a Q2 jelölésű FET végzi a motor vezérlését. Az IRFZ34N típusú FET maximális drain-source feszültsége 55 V, drain árama 29 A. Ez az alkatrész jóval nagyobb teljesítményt is képes kapcsolni, mint az általam használt motor teljesítménye, ezért a későbbiekben akár egy nagyobb motor meghajtása is lehetséges lesz vele. A FET gate lába az R5 jelölésű ellenálláson keresztül csatlakozik a mikro vezérlő RC1 lábára. A vezérlő ezen a lábán adja ki a PWM jelet. A D2 jelölésű dióda szerepe, hogy a PWM ciklus kikapcsolt periódusa alatt keletkező visszaindukált feszültség tüskéket levágja, ezek ugyanis tönkretelhetnék a vezérlő FET-et. Erre a célra egy P600M típusú diódát választottam, melynek paraméterei 1KV, 6A.

A kapcsolást úgy terveztem meg, hogy lehetséges legyen a motor áram mérése, és ennek feldolgozása a PIC mikro vezérlővel. A motor pillanatnyi áramfelvételéből meghatározható a motor pillanatnyi nyomatéka is. Az áram mérése a MAX4172 integrált kivitelű precíziós áram mérő műveleti erősítőt választottam. Ez az eszköz kiegészítve az R6 jelölésű mérő ellenállással, a kimenetén előállítja az ellenálláson átfolyó árammal arányos feszültség jelet. A 6. számú kimeneti láb a mikro vezérlő RA1-es analóg bemenetére csatlakozik. A 7. számú ~PG (Power Good) lábon logikai 0 értéket kapunk, ha a kapcsolat tápfeszültsége elérte a megfelelő működéshez szükséges feszültség szintet. Ezt a jelet a PIC RB1 digitális bemenetére kötöttem. Az R6 mérő ellenállás

kritikus része a kapcsolásnak. Ide lehetőleg minél kisebb ellenállású alkatrészt kell választani, hogy rajta minél kisebb feszültség essen. Fontos az is, hogy az ellenállás értékét minél pontosabban ismerjük, és a hőmérséklet változásának hatására ne változzon az ellenállás értéke. Mivel a motor miatt közel 3 Amper áram folyik át az ellenálláson, nagy teljesítményű eszközt kell választanunk. A szabványos értékek 10 W, 25 W, 50 W. Én 50 W ellenállást választottam a kapcsoláshoz.

2.3. Tápegység

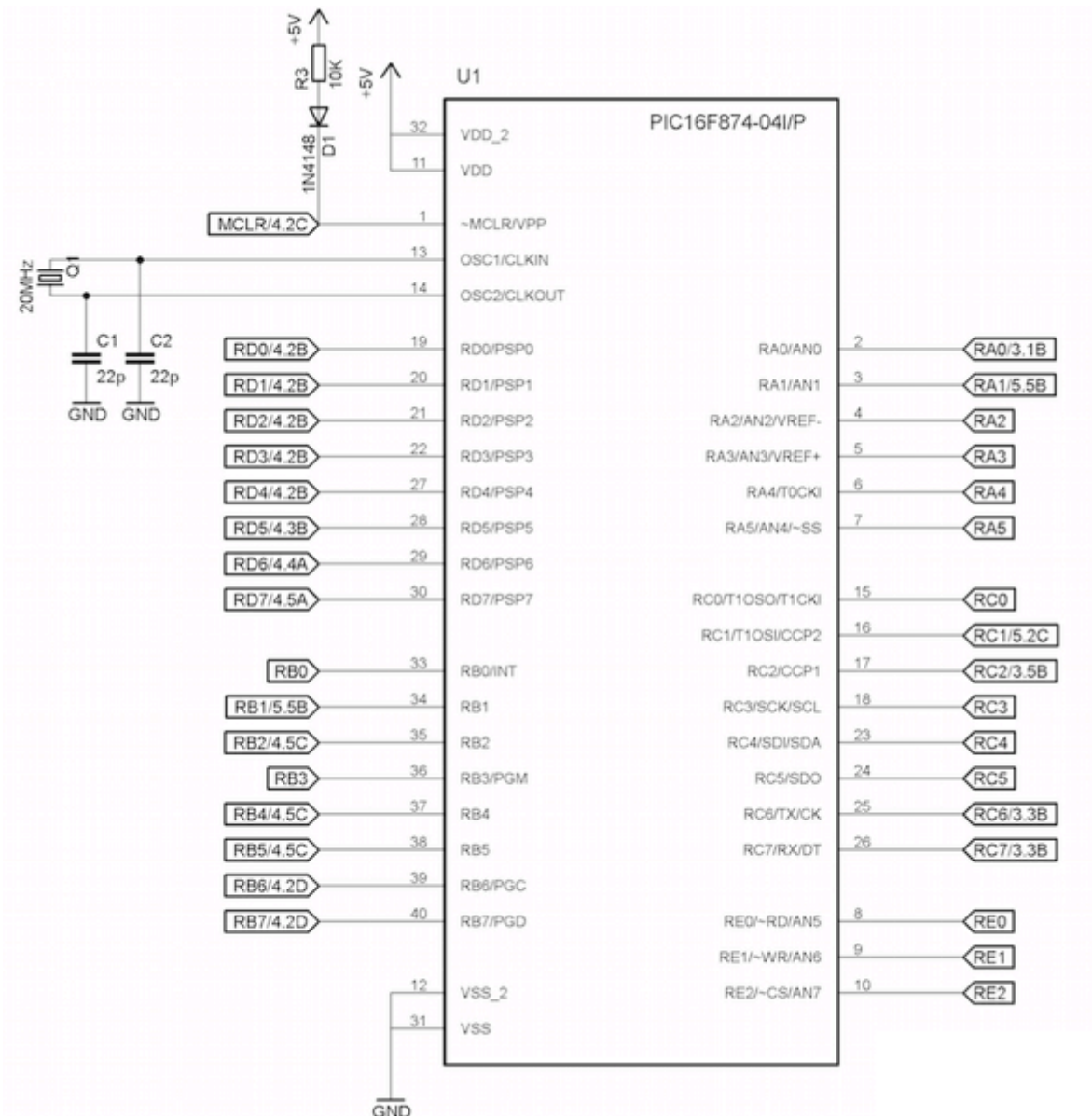


5. ábra. A tápegység kapcsolási rajza

Az eszköz működéséhez két különböző feszültség szint is szükséges. A motor 12 V DC feszültségszintről üzemel, míg a PIC és a teljes vezérlő elektronika 5 V stabilizált DC feszültséget igényel. A D10-D13 dióda híd egy integrált kivitelű KBU606G típusú egyenirányító graetz kapcsolás. Ez az eszköz 600 V csúcsfeszültség egyenirányítására képes, 6 A maximális kimeneti áramerősséggel. Mivel a motor áramfelvétele mellett a vezérlő elektronika teljesítmény igénye egy nagyságrenddel kisebb, ezért a méretezésnél a motor teljesítményét vettem alapul. A motor számára nincs szükség a feszültség stabilizálására, így a puffer kondenzátor után levehető a kapcsolásról a motor számára használt tápfeszültség. A PIC és a vezérlő elektronika számára az MC78T05C pozitív feszültség stabilizátor IC biztosítja a stabil 5.0V feszültség szintet. Az eszköz 5V és 15V közötti bemenő feszültségből képes előállítani 4% pontossággal az 5V kimenő feszültséget. Az IC képes kiszűrni a bemenő feszültség ingadozásokat, így folyamatosan stabil kimeneti feszültség

szintet produkál. Az eszköz maximális kimeneti áramerőssége 3A. A stabilizátor kimenetére bekötöttem egy LED-et az 5V feszültségszint indikálására.

2.4. Vezérlő áramkör



6. ábra. PIC16F877 kapcsolási rajza

A vezérlő elektronika legfontosabb része a PIC 16F877 típusú mikro vezérlő. Ennek a kapcsolási rajzát láthatjuk az 6. Ábrán. Ezen a rajzon a mikro vezérlő működéséhez legszükségesebb alkatrészeket helyeztem el, az összes többi periféria vezérlőhöz való csatlakozását kereszthivatkozások jelölik.

2.4.1. PIC 16F877

A PIC 16F877 a Microchip 16F középkategóriás családba tartozó 8 bites architektúrájú CMOS FLASH mikrokontrollere. Ez a vezérlő már több mint 10 éves fejlesztés, tehát nem a legmodernebb a piacon. Viszont minden funkcióval rendelkezik, amire nekem szükségem volt. Korábban, más projektekben is többször használtam ezt a típust, sok pozitív tapasztalatom volt vele.



7. *ábra.* PIC16F877 mikrokontroller

A processzor fontosabb tulajdonságai:

- 8 bites architektúra
- 35 szavas assembly nyelvű csökkentett utasítás készlet
- 200 ns ciklusidő 20MHz órajel mellett
- 8 szintes verem memória
- Watchdog timer beépített RC órajellel
- Programozható kód védelem
- Energiatakarékos SLEEP mód
- In-Circuit programozás és debuggolás 2 láb felhasználásával
- Alacsony teljesítmény igényű, gyors CMOS FLASH/EEPROM technológia
- 8K FLASH program memória
- 368 byte adat memória

- 256 byte EEPROM adat memória

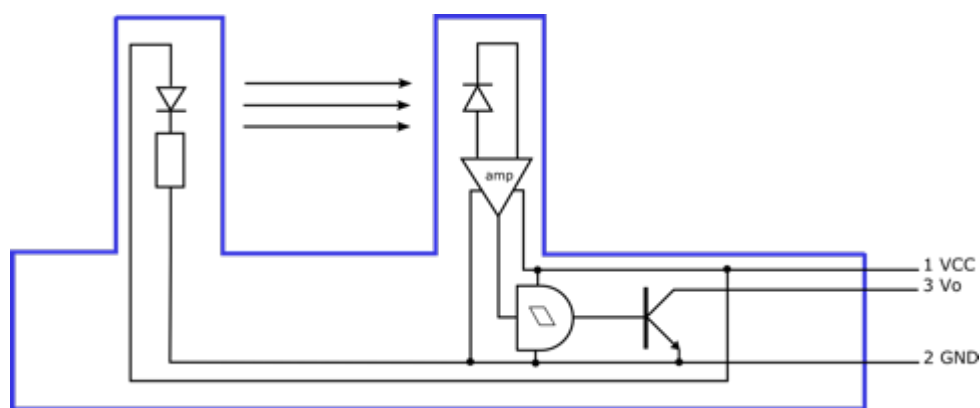
A vezérlő főbb perifériás tulajdonságai:

- 3db különálló időzítő
 - Timer 0, 8 bites időzítő/számláló, 8 bites előosztóval
 - Timer 1, 16 bites időzítő/számláló, előosztóval, külső órajellel
 - Timer 2, 8 bites időzítő/számláló, előosztóval és utóosztóval
- 2db CCP modul
 - 16 bites maximális felbontású capture modul
 - 16 bites maximális felbontású compare modul
 - 10 bites maximális felbontású PWM modul
- 10 bites 8 csatornás AD átalakító
- Szinkron soros port (SSP) SPI master móddal és I2C master/slave móddal
- USART soros kommunikációs modul
- 14 különböző perifériás interrupt
- A,B,C,D,E I/O portok

A 8. ábrán láthatjuk a mikro vezérlő blokk vázlatát. Az ábra jobb oldalán 5 darab I/O portot találunk, amikről összesen 33 darab I/O láb került kivezetésre. Láthatjuk, hogy szinte minden lábnak több funkciója van. Ezek közül az egyik a port műveletek kezelése, a másik pedig valami periféria kivezetése. Azt, hogy melyik láb, melyik funkcióját szeretnénk használni, a programból tudjuk beállítani konfigurációs regiszterek használatával. Így akár a program futása közben is megváltoztathatjuk egy láb funkcióját. A szabályozási kör legegyszerűbb implementálásához egy AD csatornára és a 2 CCP modulra volt szükségem. Az egyik CCP modul Capture módban üzemelve végzi a fordulatszám mérést, a másik pedig PWM üzemmódban állítja elő a motor vezérlő jelét. Terveztem néhány kiegészítő modult is, mint például egy LCD kijelzőből és nyomógombokból álló HMI panelt, valamint az USART modult felhasználva egy RS-232 kommunikációs modult.

2.5. Fordulatszám mérő modul

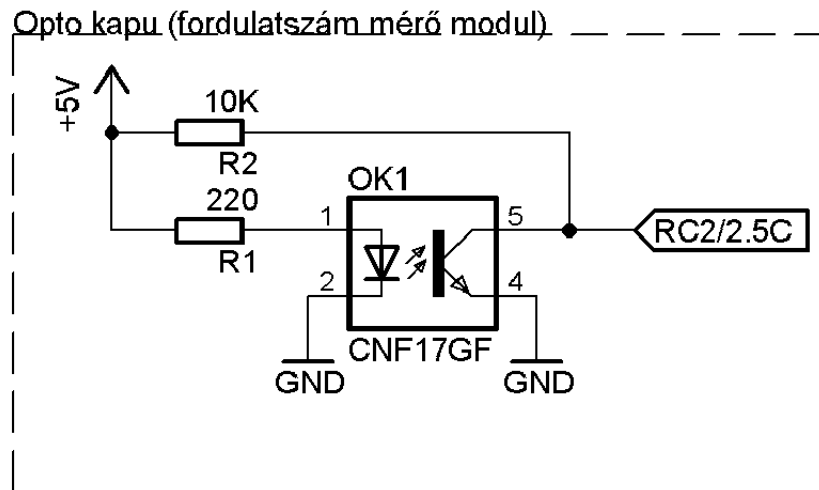
A fordulatszám mérésére egy nagyon egyszerű optokapus megoldást választottam. A motor tengelyére felszereltem egy tárcsát, amire két lyukat fúrtam, a tárcsa pereméhez közel, egymáshoz képest 180° -ra. Mivel egyelőre csak fordulatszám mérésre tervezem az eszközt, elég a kettő vagy akár egy darab furat is. Szögelfordulás és pozíció detektálásához több furatot kellene elhelyezni a tárcsa kerülete mentén. A fordulatok detektálásához egy résopto szenzort használtam fel.



9. ábra. Résopto belső felépítése

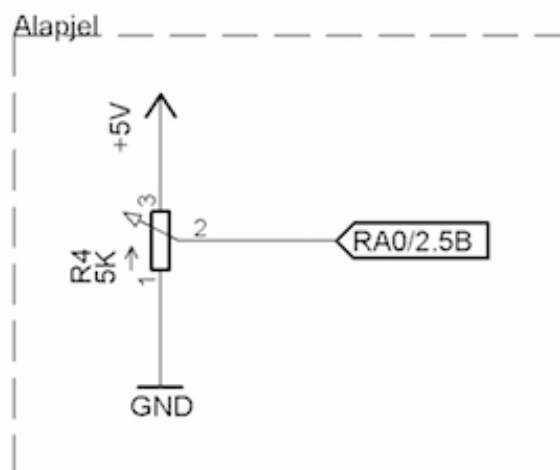
Ahogy a 9. ábra mutatja, az optokapu egyik oldalán egy infravörös fényt kibocsátó dióda található, amely, ha az eszközt tápfeszültségre kapcsoljuk, folyamatosan szolgáltatja a fényjelet. A másik oldal egy-egy fényérzékelő diódát tartalmaz, és a dióda jelét feldolgozó elektronikát. Ha a két oszlop között nincs semmilyen, a fény terjedését gátoló akadály, akkor a kimeneti kapcsoló tranzisztor kinyit. Ha viszont valami akadályt helyezünk a fény útjába, akkor a tranzisztor lezárt állapotba kerül. Tehát ha a tárcsát úgy helyezem el a két oszlop közé, hogy a furatok áteresszék az adó oldali dióda fényjelét, akkor mérni lehet a fél fordulathoz tartozó időt, amiből számolható a fordulatszám értéke.

Az 10. ábrán látható az optokapu általam használt bekötési módja. A gyártók az adatlapokban is ezt a bekötést javasolják tipikus kapcsolásként. Én a Sharp által gyártott GP1A22LC típusú eszközt használtam.



10. ábra. Fordulatszám mérő modul kapcsolási rajza

2.6. Alapjel képző modul



11. ábra. Alapjel képző modul kapcsolási rajza

Az alapjel előállítására egyszerű potenciométert használunk, amelynek értéke $5K\Omega$. A potenciométer a mikro vezérlő RA0 analóg bemenetként konfigurált lábára 0-5V közötti feszültséget kapcsol. A processzor AD átalakítója szuccesszív aproximáció elvet használva állítja elő a 10 bites digitális jelet.

2.7. Soros kommunikációs modul

Az eszközt alkalmassá kellett tenni számítógéppel való kommunikációra. Az USART modult felhasználva a PIC mikro vezérlő és egy PC között soros port-on kommunikáció hozható létre.

A soros aszinkron kommunikáció lényegesen különbözik más soros adatátviteli megoldásoktól. Nem használunk külön vezetéket, hogy az adó a vevő felé küldhesse az adatforgalom szinkronizálására szolgáló órajelet. Az aszinkron soros kommunikáció megvalósítására három vezetékre van szükségünk, amelyek közül az egyik a közös föld vezeték, a másik kettő pedig az adó és vevő vezetékek. Mivel nincs közös szinkronizáló órajel, ezért a kommunikáció sikeres lebonyolításához előre tisztázni kell mind a két oldalon a használt protokollt. Be kell állítani a közös átviteli sebességet, mivel szinkronizáló órajel híján az adó és a vevő oldalon is ugyanazt az értékű órajelet kell használni. Az átviteli sebesség (baud rate) megadja a másodpercenként átvitt bitek számát. Általános esetben a 9600 baud rate-t használjuk. Egy küldött adatcsomag 8 bitből áll, viszont a vevő oldali szinkronizációhoz szükség van egy start és egy stop jelzőbit használatára is. Használhatunk hibadetektálásra paritás bitet, ami viszont a küldött csomagok méretét 10-ről 11 bitre növeli. A paritás bit használatáról is előre meg kell egyezni a kommunikációban résztvevő feleknek.



12. ábra. Soros kommunikáció adatátvittele

A 12. ábrán láthatjuk egy adatcsomag küldését. Amíg az adó nem küld adatot, addig a vonalat logikai 1-es szinten tartja. Az adat küldés kezdetét jelentő start bit értéke mindig logikai 0. Ha a vevő azt érzékeli, hogy a vonalon a jelszint lecsökkent, vár fél bit időt, majd újra mintát vesz a vonalról. Ez körülbelül a start bit idejének a felénél következik be. Ha ekkor még mindig alacsony szinten van a vonalunk, akkor feltételezzük, hogy ez valóban egy start bit és nem valami zavar a jelvezetéken. Így a beállított baud rate-hez tartozó bitidőnként mintát veszünk a vonalról. Ha a start bitet követő 9. bit magas jelszintű, akkor a vételt befejezettnek tekintjük. Innen kezdve várakozunk

a következő start bit megérkezésére. A stop bit után akár azonnal állhat egy újabb start bit, vagy az adó tetszőleges ideig magas jelszinten tarthatja a vonalat, ha nincs közölni való információja.

Paritás bit használata esetén az adat bitek mögé bekerül egy 9. bit. Ezzel hiba detektálást lehet végezni. Ha az adó által küldött adatban a magas szintű bitek száma páros, akkor a paritás bit értékét 1-re állítja az adó. A vevő oldalon megszámloljuk a magas szintű biteket és összevetjük a kapott eredményt a paritás bit értékével. Ha ezzel a módszerrel hibásnak érzékeljük a kapott adatcsomagot, akkor a vevő oldalon nem használjuk fel, eldobjuk. Belátható azonban, hogy ez a módszer nem működik abban az esetben, ha páros számú bit sérül meg, illetve a hiba helyének detektálása sem lehetséges.

6. táblázat. A soros kommunikációs protokoll rétegei

Alkalmazási réteg	Adatcsomagok felépítésének a definiálása
Adat réteg	Átviteli sebesség, adatbitek számának, jelzőbiteknek a definiálása
Logikai réteg	A logikai 1 és 0-hoz tartozó feszültség szintek definiálása
Elektromos réteg	A használt feszültség szintek definiálása
Fizikai réteg	Vezetékek számának, csatlakozók típusának definiálása

A fent leírtakat a kommunikációs protokoll felső két rétege, az adat réteg és az alkalmazási réteg definiálja, mint azt a 6. táblázat mutatja. Mivel ezek a rétegek az adatok kezeléséről szólnak, az általuk meghatározott beállítások szoftveresek, így ezeket a kommunikációt lebonyolító programban kell beállítani mind a küldő, mind a fogadó oldalon.

Ahogy a fejezet elején írtam, a PIC és a PC között kell soros kommunikációt létrehozni. Hardveres szempontból problémát okoz azonban, hogy a két eszköz nem egyforma soros protokollt használ. A mikro vezérlő a TTL – UART protokollt ismeri, a PC viszont az RS-232-est. A két protokoll adat és alkalmazási rétege megegyezik, így szoftveres szempontból nem okoz problémát az összepárosításuk. A különbség az elektromos és a logikai rétegben keresendő.

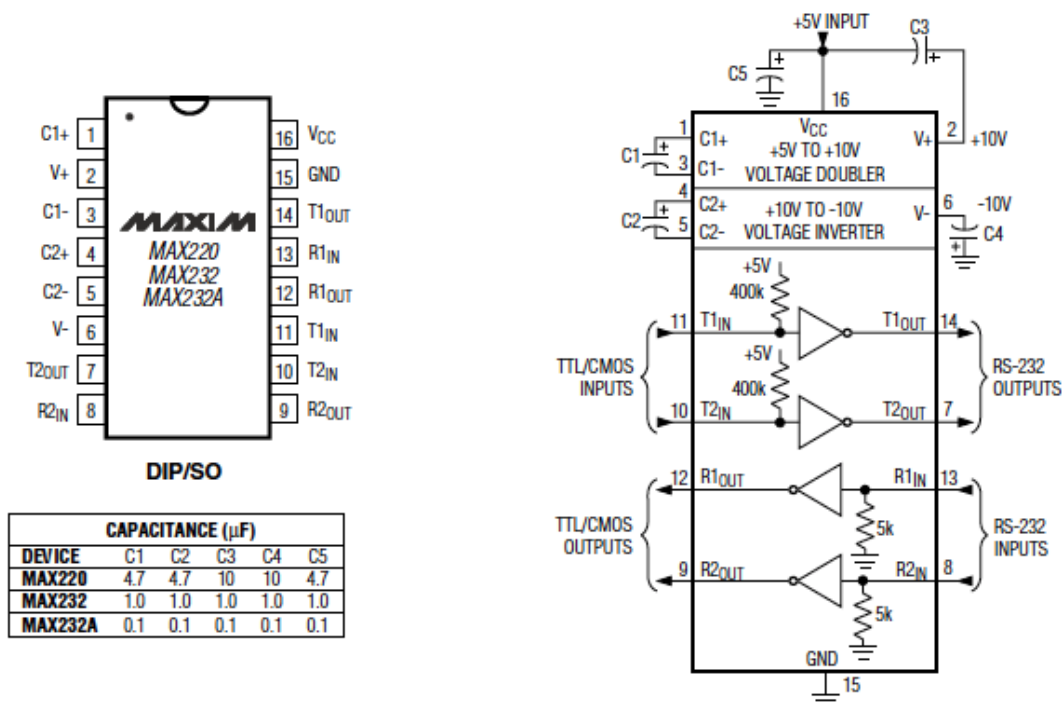
A használt feszültség szintek:

- TTL: 0V és +5V
- RS232: -25V - +25V (PC-k esetében -12V - +12V)

Logikai jelszintek:

- TTL: logikai alacsony szint 0V, magas szint +5V
- RS232: logikai alacsony szint +3V és +25V között, magas szint -3V és -25V között. -3V és +3V között tiltott sáv van amit a kommunikáció során nem használhatunk.

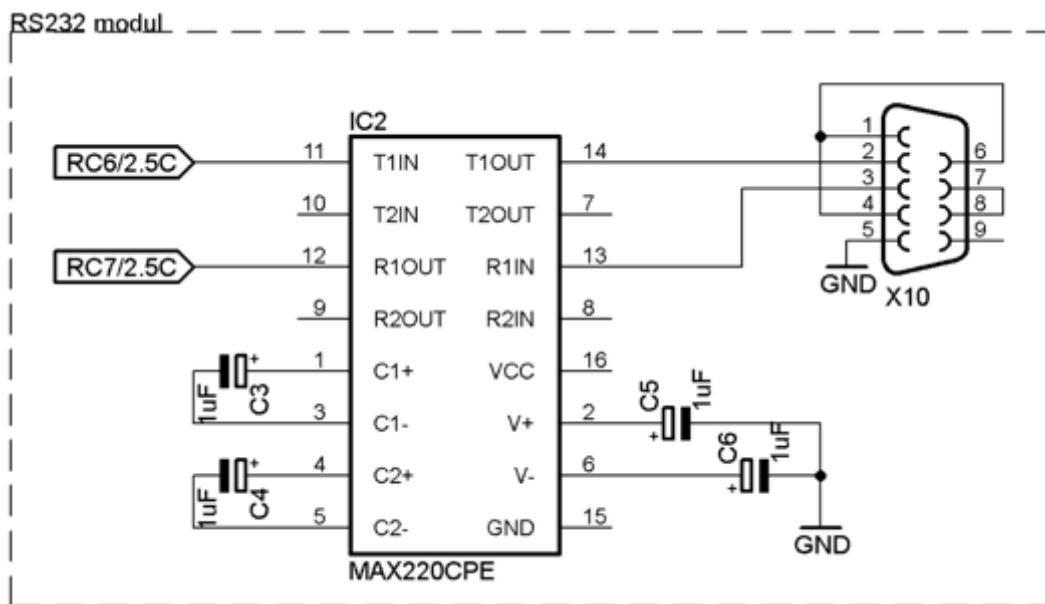
Ahhoz, hogy a két különböző protokollt használó eszközt összeköthessük, egy szint-illesztő áramkört kell használnunk. Erre a feladatra alkalmas eszköz a MAX232 IC.



13. ábra. MAX232 IC lábkiosztása és belső felépítése (forrás: [9])

Az eszköz belső felépítését a 13. ábra mutatja. Az IC +5V tápfeszültségről üzemel, ebből állítja elő az RS-232 oldal számára a feszültség szintet, az integrált feszültség konverter és inverter segítségével. Az IC-be nem tudták integrálni a konverter és inverter működéséhez szükséges kapacitásokat, így azokat kívülről kell a megadott lábakra illeszteni. A 13. ábrán található táblázatból kiolvashatjuk, hogy a MAX232 típushoz 1 uF értékű kapacitásokat kell használnunk.

Az általam alkalmazott bekötést az 13. ábrán láthatjuk. Az IC alkalmas lenne 2 db kommunikációs csatorna illesztésére is, nekünk viszont csak egy csatornára van szükségünk, ezért a T2 és R2 lábak üresen maradnak. A PIC az RC6/TX lábán képes adni, így ezt a lábát a szintillesztőn keresztül a PC oldali sub-D9 csatlakozó 2. lábára kötjük, ez ugyanis az RS-232 szerinti vevő láb. A PC a 3. lábon tud adni, így ezt a szintillesztőn keresztül a PIC RC7/RX vevő lábával kötjük össze. Mivel külön vonalon megy az adás, illetve a vétel, belátható, hogy full duplex kommunikációról beszélünk. A kilenc pólusú csatlakozó fennmaradó lábait például PC – modem kommunikáció esetén használják különböző handshake-ek lebonyolítására. Mi ezeket nem használjuk fel, ezért a PC oldali csatlakozóban a képen látható átkötéseket kell kialakítani.



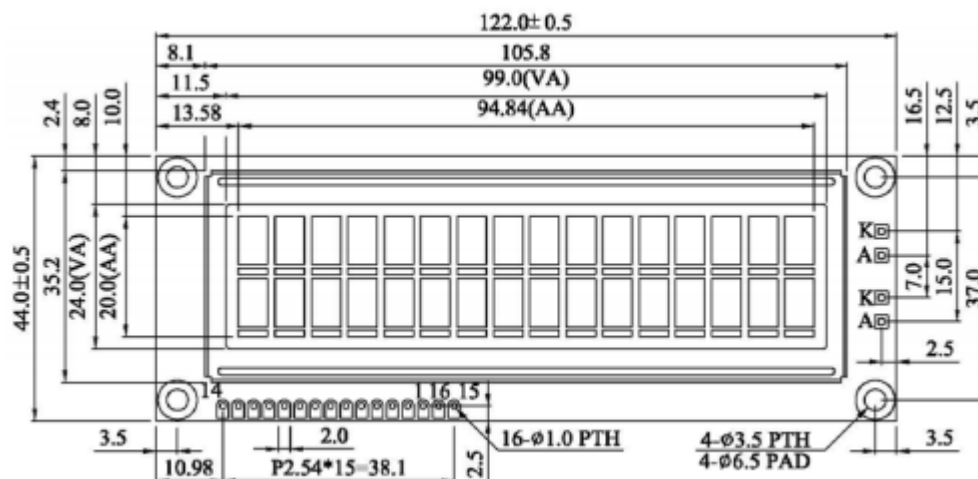
14. ábra. Soros kommunikációs modul, kapcsolási rajza

2.8. HMI modul

A felhasználó és az eszköz közötti információcsere megvalósítására egy HMI modult is terveztem az eszközhöz. A modul tartalmaz egy két soros, 16 karakteres LCD modult, három nyomógombot, és egy duo LED-et. Ide sorolható még az alapjel képzésére szolgáló potenciométer is, hiszen az is egy beviteli eszköz. A kijelzőn alapértelmezetten a beállított alapjel értéke jelenik meg, valamint a mért fordulatszám. Így vizuális úton kapunk visszajelzést arról, hogy mit állítottunk be a potenciométeren, valamint, hogy a szabályozónak sikerült-e a motor fordulatszámát a kívánt értékre állítania. Az LCD és a nyomógombok segítségével viszont lehetőség nyílik arra is, hogy létrehozzunk a programban egy menü struktúrát, amelyben elhelyezhetünk különböző beállításokat, üzemmódokat. Például az LCD és a nyomógombok segítségével kiválaszthatjuk, hogy milyen típusú szabályozási algoritmust futtasson az eszköz, vagy beállíthatjuk a szabályozó paramétereit is.

2.8.1. Karakteres LCD kijelző

Az általam felhasznált kijelző a HD44780 LCD meghajtó processzorra épül. Ez a vezérlő típus a legelterjedtebb a karakteres LCD-k között.



15. ábra. az LCD (forrás: [10])

A 15. ábrán látható az LCD nem méretarányos, előlnézeti rajza. A kijelző a karaktereket két sorban képes megjeleníteni, soronként 16 darabot. A csatlakozók megnevezéseit és funkcióit a 7. táblázat tartalmazza.

Az 1. és 2. csatlakozókra kötjük a tápfeszültséget. A 3. csatlakozóra kapcsolt feszültséggel lehet beállítani a kijelző kontrasztját. Építhetünk két ellenállásból egy fix feszültség osztót is, de amíg nincs tapasztalatunk arról, hogy mekkora osztó szükséges az ideális kontraszt beállításához, addig inkább egy trimmer potenciométert érdemes beszerezni, aminek értéke 5 K Ω . A változtatható ellenállás használata olyan esetekben is előnyös, amikor a kijelzőt változó fényviszonyok között szeretnénk használni, hiszen így később is lehetőségünk nyílik a kontraszt beállítására. A potenciométer bekötési módját a 16. ábrán láthatjuk.

Szintén ezen az ábrán megfigyelhető, hogy az R/W láb a föld potenciálra van kötve. HD44780 vezérlővel rendelkező kijelzők read és write módban működnek. Write módban parancsokat küldhetünk az adatbuszon keresztül, illetve ebben az üzemmódban tudunk karaktereket küldeni a kijelzőnek megjelenítésre. Read üzemmódban az LCD státusz jelzéseket (busy flag) tud küldeni az adatbuszon. Ezek a jelzések például arra vonatkoznak, hogy a kijelző vezérlője feldolgozta-e már az előzőleg küldött adatot. Ennek az üzemmódnak a használata azonban nem elterjedt, inkább időzítéseket szokás használni a busy flag figyelése helyett. Ha az RW láb konstans

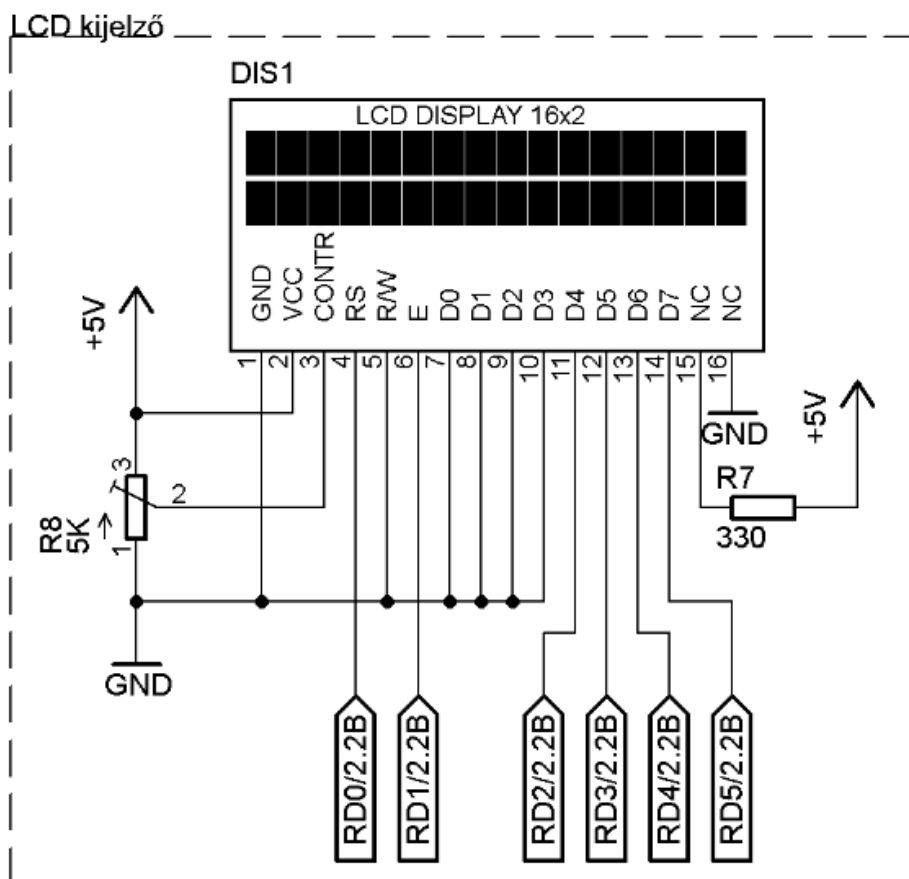
alacsony szintű jelet kap (föld potenciálra kötjük), a kijelző folyamatosan write állapotban lesz.

7. táblázat. Az LCD csatlakozópontjainak funkciói

Csatlakozó száma	megnevezése	funkciója
1	Vss	Föld potenciál
2	Vdd	Pozitív tápfeszültség
3	Vee	Kijelző kontraszt
4	RS	Register select
5	R/W	Read / write
6	E	Enable
7	D0	1. Adatbit
8	D1	2. Adatbit
9	D2	3. Adatbit
10	D3	4. Adatbit
11	D4	5. Adatbit
12	D5	6. Adatbit
13	D6	7. Adatbit
14	D7	8. Adat bit
15	A	háttérvilágítás LED anód
16	K	háttérvilágítás LED katód

Az RS lábat (register select) a mikro vezérlő egyik kimenetére kell bekötni. Ennek a lábnak a segítségével adhatjuk meg, hogy adatot küldünk a kijelzőnek vagy pedig parancsot. Ha az RS láb logikai értéke 1, akkor a kijelző a küldött adatot karakterként értelmezi, ha pedig logikai 0 az értéke, akkor parancsként értelmezi a mikro vezérlő által küldött adatot.

Az E (enable) láb funkciója, hogy a D0-D7 adatbuszra beállított értékek adatátvitelét elindítsa. Az átvitel megkezdéséhez egy minimum 230 ns idejű impulzust kell adni az E lábra. A parancs kiadás ciklusideje minimum 500 ns.



16. ábra. LCD kijelző, kapcsolási rajz

A D0-D7 lábak az adatbevitelre szolgálnak. Ezek a típusú kijelzők kétféle adatátviteli módban képesek működni, 4 bites és 8 bites módban. 4 bites üzemmódban a felső 4 bitet használjuk adatátvitelre, ezért ahogy ez a 16. ábrán is látszik, az alsó 4 bithez tartozó lábakat föld potenciálra kötöttem, így ezek logikai értéke folyamatosan 0. A 4 bites mód beállításához szükség van a bekapcsolás után egy konfiguráló parancsra, amelyre részletesebben később térek ki. Ennek az üzemmódnak a használata mikro vezérlős kapcsolásoknál lehet hasznos, mivel így spórolhatunk az I/O lábakkal. Igaz, hogy így a kijelzőre írás sebessége felére csökken, mivel minden adatot két 4 bites részletben kell elküldeni, de ez a sebességcsökkenés szemmel nem észrevehető, mivel egy adatsomag küldési ideje kb. 50 us.

Ha a kijelző háttérvilágítását is szeretnénk használni, akkor a 15. anód lábat egy 330 Ω -os ellenálláson keresztül a tápfeszültségre kötjük, és a 16. katód lábat pedig föld potenciálra. Így a tápfeszültség bekapcsolása esetén folyamatos háttérvilágítást kapunk. Ha szeretnénk kapcsolhatóvá tenni a háttérvilágítást, beköthetjük az anódot a mikro vezérlő egyik kimenetére is.

8. táblázat. kijelző konfigurációs parancsok

Parancs	D7	D6	D5	D4	D3	D2	D1	D0
Kijelző törlése	0	0	0	0	0	0	0	1
kijelző és kurzor alaphelyzetre	0	0	0	0	0	0	1	-
Karakter beviteli mód	0	0	0	0	0	1	I/D	SH
Kijelző be/ki	0	0	0	0	1	D	U	B
Kurzor és eltolás	0	0	0	1	S/C	R/L	-	-
Funkciók beállítása	0	0	1	DL	N	F	-	-
CGRAM cím beállítása	0	1	A	A	A	A	A	A
DDRAM cím beállítása	1	A	A	A	A	A	A	A

Az 8. táblázatban foglaltam össze a kijelző konfigurálásához használható parancsokat. A táblázatban szereplő bit jelölések magyarázata:

- I/D: Kurzor mozgatás, 1=növelés, 0=csökkentés
- SH: Kijelző eltolás engedélyező bit
- D: Kijelző be/ki kapcsolása
- U: Kurzor aláhúzás be/ki
- B: Kurzor villogtatása be/ki
- S/C: Kijelző eltolás és kurzor mozgatás kiválasztó bit, 1=kijelző eltolás, 0=kurzor mozgatás
- R/L: Eltolás iránya, 1=jobbra tol, 0=balra tol
- DL: Üzem mód beállítása: 1=8 bites mód, 0=4 bites mód
- N: Sorok számának beállítása: 1=2 soros mód, 0= soros mód
- F: Karakter méret beállítása: 1=5/10 pixel, 0=5*7 pixel
- A: Adatbit

Ahhoz, hogy az LCD-nek az 8. táblázatban összefoglalt parancsokat elküldhessük, az RS lábat logikai 0 értékre kell állítani.

Karakterek kiírásához először az RS lábat logikai 1 értékre kell állítani, így az LCD a D0-D7 lábakon elküldött adatot karakterként fogja értelmezni, nem pedig parancsként. A 17. ábrán láthatjuk a HD44780 vezérlő által ismert karaktereket. A táblázat bal oldali oszlopa tartalmazza a karakterekhez tartozó kódok alsó 4 bitjét, a felső sor pedig a kód felső 4 bitjét. A karakterkészlet tartalmazza az angol abc kis és nagy betűit, 0-tól 9-ig a számokat, valamint az írott szövegben és

informatikában leggyakrabban használt írásjeleket. A táblázat jobb oldalán láthatjuk, hogy a vezérlő karakterkészlete tartalmazza a japán betűk egy részét is, valamint néhány speciális ékezetes karaktert és görög betűket is.

Lower 4 Bits	Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	a	P	`	P				-	9	3	α	p	
xxxx0001	(2)			!	1	A	Q	a	9			•	7	4	ä	q	
xxxx0010	(3)			"	2	B	R	b	r			「	イ	ツ	×	β	θ
xxxx0011	(4)			#	3	C	S	c	s			」	ウ	テ	モ	ε	ω
xxxx0100	(5)			\$	4	D	T	d	t			、	エ	ト	ホ	μ	Ω
xxxx0101	(6)			%	5	E	U	e	u			•	オ	ナ	1	ç	Ü
xxxx0110	(7)			&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)			'	7	G	W	g	w			7	†	ヌ	ラ	g	π
xxxx1000	(1)			(	8	H	X	h	x			ィ	ク	ネ	リ	5	Σ
xxxx1001	(2)			)	9	I	Y	i	y			ウ	7	ル	ル	ˆ	y
xxxx1010	(3)			*	:	J	Z	j	z			エ	コ	ン	レ	j	7
xxxx1011	(4)			+	;	K	L	k	l			オ	サ	ヒ	ロ	*	5
xxxx1100	(5)			,	<	L	¥	1	l			ホ	シ	フ	ワ	φ	π
xxxx1101	(6)			-	=	M	J	m	j			ユ	ズ	ン	ン	÷	÷
xxxx1110	(7)			•	>	N	^	n	+			ヨ	セ	ホ	°	ñ	
xxxx1111	(8)			/	?	O	_	o	+			ッ	リ	マ	°	ö	■

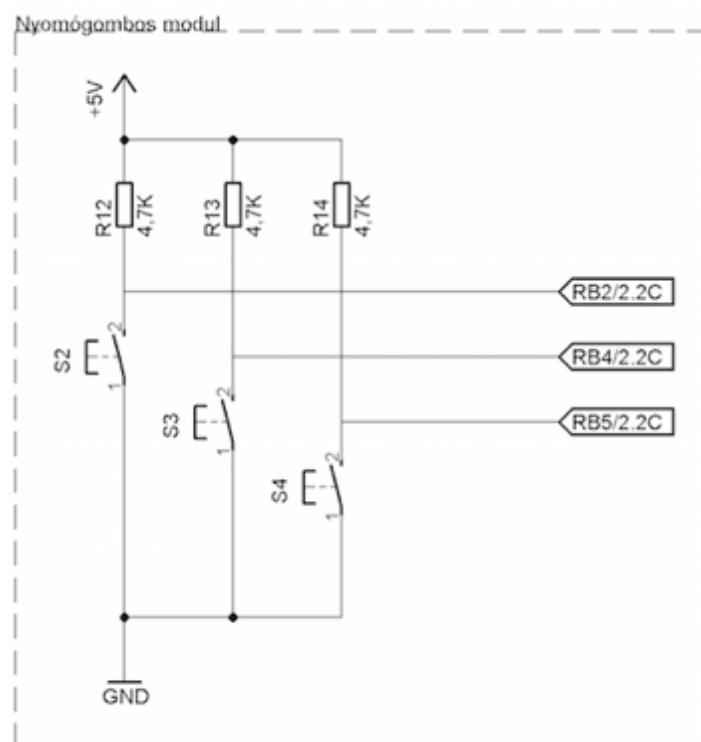
17. ábra. A HD44780 LCD vezérlő karakter készlete (forrás: [10])

A karakterek előhívásához használt bináris számok valójában az adott karakterekhez tartozó ASCII kódok. Ha a szoftverből szeretnénk adatot kiíratni a kijelzőre, akkor használhatjuk a char adattípust az elküldeni kívánt karakter megadásához.

2.8.2. Beviteli eszköz

A HMI panelek (Human Machine Interface) kapcsolatot teremtenek a felhasználó és a használt eszköz között. Feladatuk, hogy információt tudjanak átadni a kezelő személynek, valamint, hogy rajtuk keresztül a kezelő utasításokat adhasson az eszköz felé. Ennek érdekében az LCD kijelző mellé nyomógombokat is elhelyeztem a panelen. Az általam tervezett három nyomógomb navigációs célt szolgál. Ha az LCD felhasználásával létre szeretnénk hozni egy menü struktúrát, akkor a nyomógombok segítségével tudunk a menüben navigálni. Ilyen felhasználás esetén a gombok például a következő funkciókat kaphatják: FEL, LE, ENTER.

Egy 4x4 nyomógombos hexa billentyűzet alkalmazásával könnyebben kezelhető eszközt valósíthatnánk meg. A későbbiekben, igény esetén van lehetőség 4x4-es billentyűzet illesztésére, mert a mikrovezérlő még rendelkezik több mint nyolc üres lábbal.



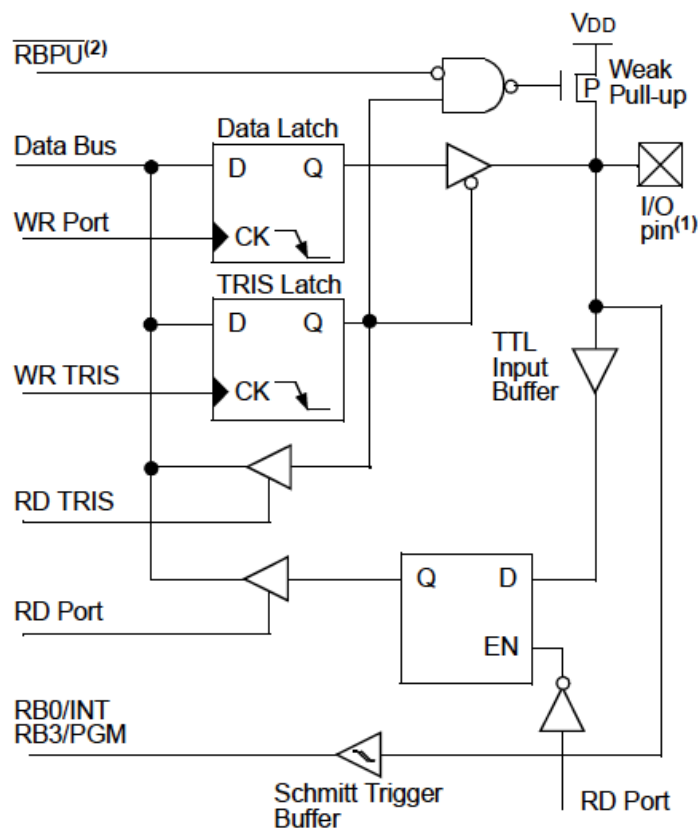
18. ábra. Nyomógombok, kapcsolási rajz

Ha a mikrovezérlő egy lábát bemenetként szeretnénk használni, nem elég az, ha a tápfeszültséget egy nyomógombon vagy kapcsolón keresztül az adott lábra kötjük. Ebben az esetben, ha a kapcsoló nyitott állapotban lenne, akkor a láb nem lenne sem a föld potenciálra sem pedig a tápfeszültségre kötve, vagyis a logikai értéke nem lenne se 0 sem pedig 1, vagyis egy harmadik úgynevezett

nagyimpedanciás állapotba kerülne. Ebben az állapotban a láb lebeg, vagyis nem lehet megállapítani a logikai értékét, a lábra ható mágneses és kapacitív zavarok függvényében mozog a két logikai állapot között. Belátható, hogy ez a módszer nem vezetne stabil működéshez. Ha megfelelően szeretnénk eljárni, és azt akarjuk, hogy a bemenetünk működése stabil legyen a 18. ábrán látható módon kell a nyomógombokat bekötni. Alaphelyzetben nyitott kapcsolók esetén a tápfeszültség a 4.7K Ω -os ellenállásokon keresztül a PIC bemeneti lábaira kerül. Ezeket az ellenállásokat felhúzó ellenállásoknak nevezzük, mert szerepük a felhúzó áram értékének beállítása. 5V tápfeszültség esetén így a felhúzó áram 1.063 mA. Egy kis kitérővel nézzünk most meg egy másik megoldást is. A PIC 16F877 mikrovezérlő B portjának minden lába rendelkezik belső programozható felhúzó ellenállással. Ezek használata esetén a külső felhúzó ellenállások alkalmazása elhagyható, kevesebb alkatrészt kell felhasználnunk. Belső felhúzó ellenállások használata esetén a nyomógombot a föld potenciál és a bemeneti láb közé kell bekötni. Az 19. ábrán látható a PORTB RB3-RB0 lábak belső felépítése. A felhúzó ellenállás egy áramgenerátor, amely katalógus szerint 50-400 μ A felhúzó áramot biztosít. A PIC 16F877 belső felhúzó ellenállás értéke körülbelül 20K Ω . A PORTB belső felhúzó ellenállásainak engedélyezése az option regiszter RBPU bitjének törlésével történik. Ennek a bitnek a törlése viszont a PORTB összes lábán engedélyezi a felhúzó ellenállásokat. Mivel én csak három bemeneti lábon használok nyomógombokat, ezért nem használtam a belső felhúzó ellenállásokat.

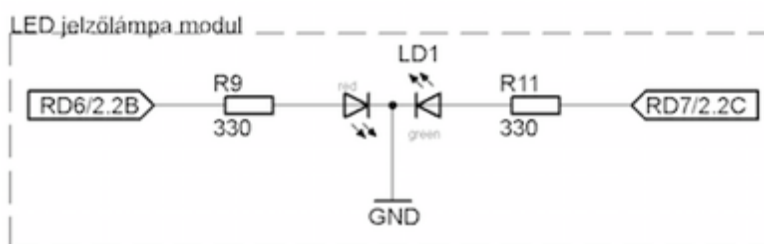
Térjünk vissza az általam használt kapcsolás működésének elemzéséhez. Azt már láthattuk, hogy nyitott kapcsoló esetén a bemeneti láb logikai magas szintre kerül. Ha a kapcsolót zárjuk, a PIC lábát föld potenciálra húzzuk, tehát állapota logikai alacsony lesz. Belátható tehát, hogy ez a kapcsolás valóban mindig stabil állapotba hozza a bemenetünket. Azt is észrevehetjük, hogy a kapcsoló fordított logika szerint működik, tehát zárt állapotban lesz a bemenet 0 állapotban. Erre programozás közben kell figyelemmel lenni.

Mikrovezérlős alkalmazásokban nyomógombok használatakor könnyen szembesülhetünk a pergés jelenségével. Egy nyomógomb megnyomásakor az általa kapcsolt feszültség jelalakja nem egy szabályos felfutó élet mutat, mint ahogy azt várnánk, hanem stabilizálódás előtt többször „ugrál” az alacsony és magas értékek között. Ha egy bemenet értékét szoftveresen figyeljük, akkor a mikrovezérlő egy gombnyomást úgy érzékel, mintha többször is megnyomtuk volna a gombot, mivel valóban több bemeneti érték változás is történt. Ez a jelenség tehát hibás szoftver működést eredményezhet. A jelenség kivédhető hardveresen is, egy RC kör vagy egy flip/flop áramkör használatával, illetve szoftveresen is. Ennél a projektnél a takarékosabb szoftveres megoldást alkalmaztam.



19. ábra. RB3-RB0 lábak blokk diagramja (forrás [7])

2.8.3. LED modul



20. ábra. duoLED, kapcsolási rajz

A HMI panelra az egyszerű állapotok jelzésére egy duo LED-et terveztem. Ennek a bekötését a 20. ábrán láthatjuk. Ennek a LED-nek három kivezetése van. A katód láb közös, és egy tokon belül egy zöld és egy piros LED-et tartalmaz. A vezérlésükre az RD6 és RD7 kimenetként konfigurált lábakat használom. A duoLED segítségével négy állapotot tudok kijelezni, ezeket a 9. táblázatban foglaltam össze.

9. táblázat. LED által jelzett állapotok

RD6	RD7	Szín	Állapot
0	0	-	Kikapcsolt állapot
1	0	Piros	Hibás működés
0	1	Zöld	Megfelelő működés
1	1	Narancssárga	Túl/alul szabályozás

3. Nyomtatott áramkör tervezése

Az előző fejezetekben bemutatott hardver egységeket NYÁK lapokon szereltem össze. Ezek tervezésére a Sprint layout PCB tervező szoftvert használtam. Kézenfekvő lett volna a kapcsolási rajzok elkészítésére is használni EAGLE Layout szoftver PCB tervezőjével elkészíteni a NYÁK terveket, viszont én több panelre szerettem volna szétosztani a teljes kapcsolást, ezt pedig EAGLE-ben nem tudtam volna kivitelezni.

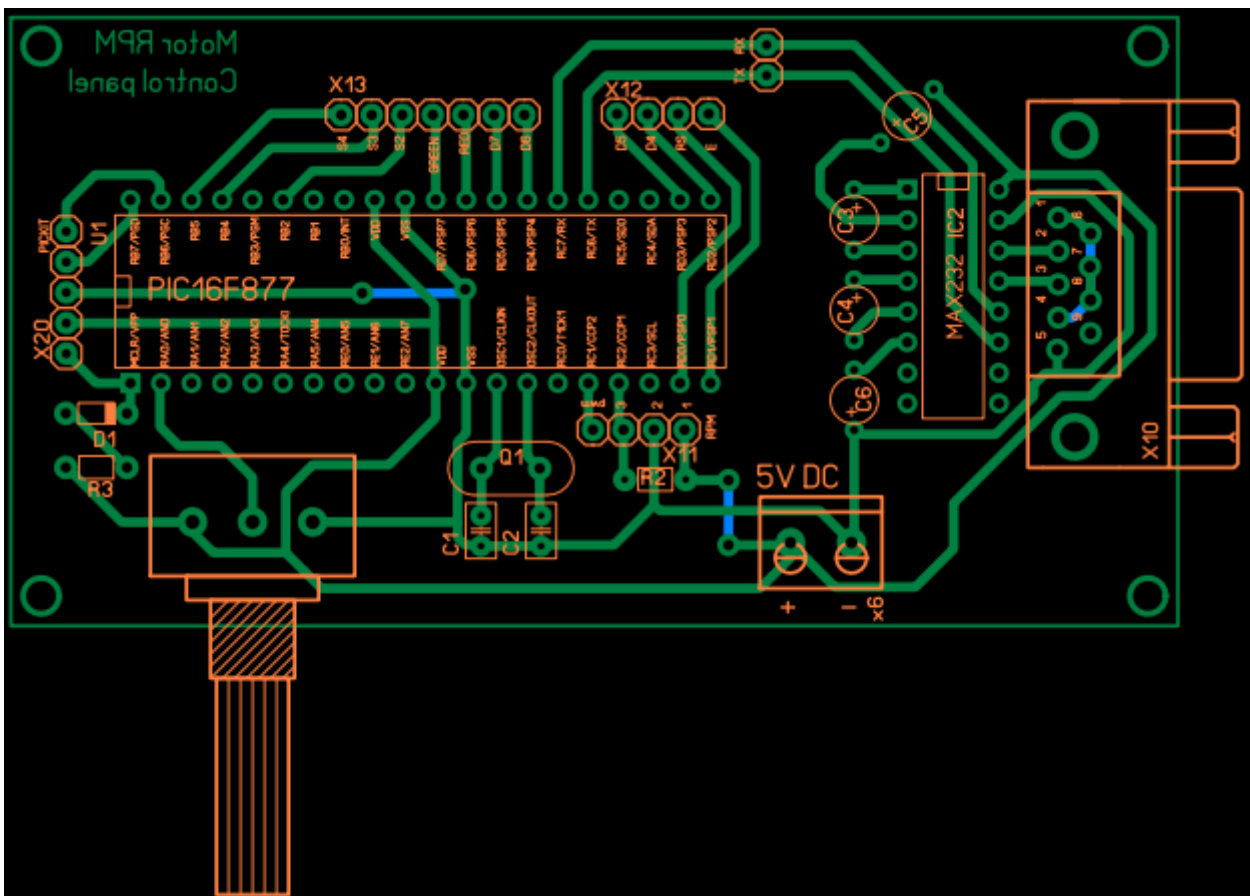
3.1. A tápegység NYÁK terve

A 26. ábrán látható a tápegység NYÁK rajza. A paneleket egy oldalásra terveztem. Zöld színnel láthatók a réz vezető sávok, ezek a panel alsó oldalán kerülnek kialakításra. A narancssárga színű rajzok a panel felső oldalára kerülő alkatrész beültetést segítő ábrák.

A NYÁK bal oldalán található a 12V AC betáplálás, jobb oldalon pedig a DC stabilizált 5V és a DC 12 V sorkapcsai kaptak helyet. A vezető sávok 2mm szélesek a tápegység és a motor hajtás panelokon, mivel ezeken nagyobb áramok folynak. A furatok mérete alkatrészenként változó, ezeket az adott alkatrészsre méreteztem.

3.3. A vezérlő panel NYÁK terve

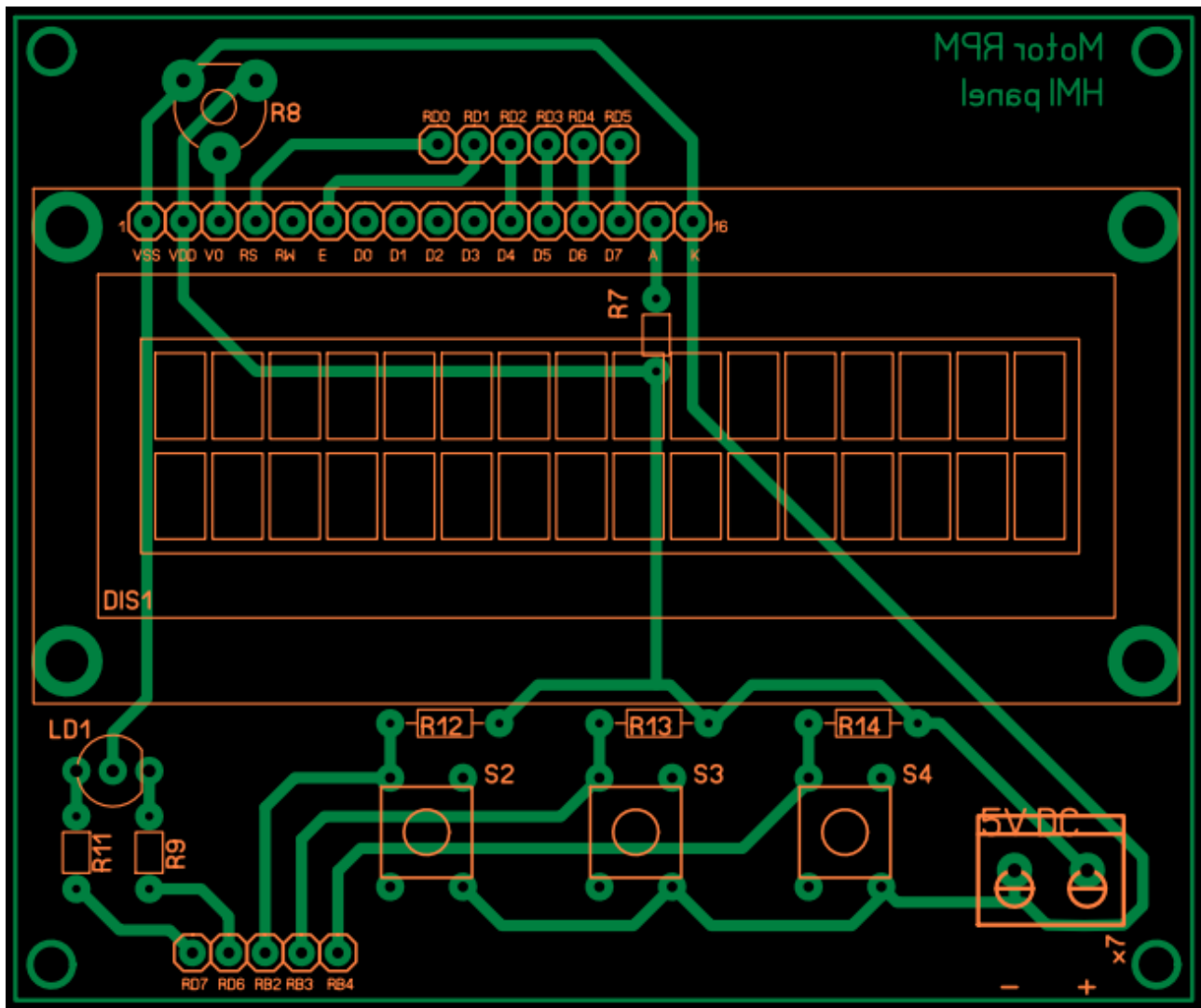
A vezérlő panelre a mikrovezérlőn kívül csak a kommunikációs modul és az alapjel beállító potenciométer került fel, illetve a PIC működéséhez szükséges alkatrészek. A többi panel jumper tükörsorok segítségével csatlakoztatható a vezérlőhöz. Ezen a panelen minden vezetősáv 0,8mm széles, mivel itt csak 5V feszültségű kis áramerősségű vezérlő jeleket használok. A sávok szélességét gyártástechnológiai okok miatt nem választottam vékonyabbra.



28. ábra. a vezérlő panel NYÁK rajza

3.4. A HMI panel NYÁK terve

A HMI panelre került minden olyan eszköz amely adatok megjelenítésére illetve bevitelére szolgál. Itt található az LCD modul és a működéséhez szükséges alkatrészek, a nyomógombok és a felhúzó ellenállásaik, és a duoLED az ellenállásaival együtt. A HMI panel külön tápellátással rendelkezik, nem a vezérlő paneltől kapja a tápfeszültséget.



29. ábra. a HMI panel NYÁK rajza

4. A mikrokontroller szoftvere

A dolgozatom eddigi részében a hardver tervezéséről és működéséről írtam, a most következő részben szeretném bemutatni a mikrokontrollerre megírt szoftvert. A hardveres részhez hasonlóan most is modulonként szeretném bemutatni a teljes programot. A vezérlő szoftvert a Microchip által fejlesztett MPLAB X IDE integrált fejlesztői környezetben írtam, és a kontroller programozását is ebből a programból végeztem. A szoftvert C nyelven írtam, és a Microchip XC8 compilert használtam a fordításhoz.

4.1. A szoftver felépítése

A legtöbb modern programozási nyelv, mint az általam használt C nyelv is, a függvények használatára épül. A függvények olyan kódrészletek amelyek akkor futnak le, ha a kódunkban meghívjuk őket. Minden C program tartalmaz legalább egy függvényt, ez pedig a main függvény. A main függvény minden C program belépési pontja, csak egyszer kerül meghívásra a program indításakor. A main függvényen belül meghívhatunk további függvényeket is, amelyek kódját mi írjuk meg. Egy függvénynek a változókhoz hasonlóan van típusa, neve, és függvény híváskor átadhatunk neki értékeket. Ezen kívül a függvényeknek lehet visszatérési értékük, amit lefutásuk után átadnak a programnak. Az x. ábra egy egyszerű függvény felépítését és meghívását mutatja. Ez a program létrehoz egy c változót amelyben eltárolja 2 és 3 szorzatát. A szorzást a „szoroz” függvény végzi el.

```
1  int szoroz (int a, int b)
2  {
3      return a * b;
4  }
5
6  int main (void)
7  {
8      int c;
9      c = szoroz(2,3);
10 }
```

35. ábra. Függvény hívás

A függvények tehát lehetővé teszik számunkra egy visszatérő feladat egyszerű megoldását, a kód részlet többszöri felhasználását. A fenti példánál maradva, ha a programban többször kell szorzást

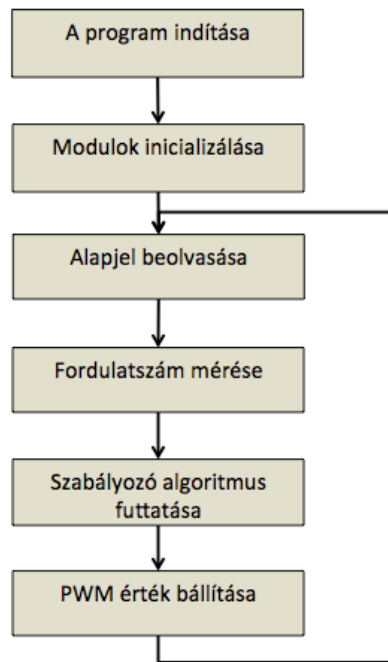
végezni, elegendő egyszer megírni a szorzás műveletét. Ezután a „szoroz” függvény bármennyiszer meghívható, különböző értékekkel. A függvények csak egyszer foglalják le a helyet a memóriában, így ha egy függvényt a program futása során sokszor meghívunk, jelentős mennyiségű program memóriát takaríthatunk meg. Ezen kívül a függvények használata rendezett és könnyen átlátható kódot eredményez.

A programom felépítésekor nemcsak függvényeket használtam a rendszerezésre, hanem a különböző program modulokat külön forrásfájlokban írtam meg. Az MPLAB X és más modern fejlesztői környezetek projektekben dolgoznak. Így lehetővé teszik azt, hogy egy program kódját külön forrásfájlokban helyezze el a programozó, majd ezeket a fájlokat a fordító egyben fordítsa le. Ez a megoldás kényelmesen kezelhető, könnyen átlátható, újra felhasználható és elegáns megjelenésű kódot eredményez.

Az általam használt projekt struktúra a következő:

- `main.c`: Ez a file tartalmazza a `main` függvényt és a program fő vezérvonalát.
- `ad_modul.c`: Az AD modul inicializáló függvényét tartalmazza, valamint az alapjelet beolvasó függvényt.
- `rpm_modul.c`: A CCP modul capture üzemmódját inicializáló függvényt tartalmazza, valamint a fordulatszám mérő függvényt.
- `control_modul.c`: A szabályozó algoritmusok függvényeit tartalmazza.
- `pwm_modul.c`: A CCP modul PWM üzemmódját inicializáló függvényt tartalmazza, valamint a pwm jelet előállító függvényt.
- `usart_modul.c`: A soros kommunikációhoz szükséges függvényeket tartalmazza.
- `lcd_modul.c`: Az LCD használatához szükséges függvényeket tartalmazza.
- `io_list.h`: A ki és bemenetek szimbólumait deklaráló makrókat tartalmazza.
- `signatures.h`: A fent bemutatott fájlokban található függvények fejléceit tartalmazza.
- `lcd_modul.h`: Az LCD használatához szükséges függvények fejléceit tartalmazza, valamint az LCD használatához szükséges makrókat.
- `usart_modul.h`: A soros kommunikációhoz szükséges függvények fejléceit tartalmazza.

A szoftver működésének blokkvázlatát a 36. ábrán láthatjuk. A következő fejezetekben a fent leírt forrásfájlok tartalmát és a blokkvázlat elemeit fogom részletesen bemutatni. Először bemutatom egyenként a különböző modulokat, majd végül az ezeket összefogó és vezérlő fő programról írok.



36. ábra. A szoftver blokkvázlata

4.2. Az alapjel feldolgozó A/D modul

Az alapjel beállítására hardveres szempontból egy potenciométer szolgál. A potenciométer által szolgáltatott analóg feszültség jelből állítja elő a szoftver a fordulatszám alapjelet a szabályozó algoritmus számára. Az alapjel feldolgozással és az AD átalakítóval kapcsolatos függvényeket az `ad_modul.c` fileban írtam meg.

A PIC16F877 mikrovezérlő AD modulját használat előtt konfigurálni kell. Erre szolgálnak az `ADCON0` és az `ADCON1` regiszterek.

Amint azt az *10. táblázatban* láthatjuk, ez a regiszter használható a modul bekapcsolására, a konverzió indítására, a csatorna és az órajel kiválasztására.

Az `ADCON1` regiszter 0-3 bitjei az analóg port konfigurálásra szolgálnak. Ezzel a 4 bittel állíthatjuk be, hogy az analóg port melyik lábát használjuk analóg bemenetként, és melyiket digitális ki/bemenetként. Ezen kívül vannak olyan beállítási lehetőségek is, amelyek használatával az analóg port bizonyos lábait referencia feszültség megadására használhatjuk. Az *11. táblázatban* láthatjuk a beállítási lehetőségeket.

10. táblázat. ADCON0 regiszter

Bit 0	ADON	AD modul bekapcsoló bit	0	AD kikapcsolva	
			1	AD bekapcsolva	
Bit 1	-				
Bit 2	GO/ <u>DONE</u>	AD konverzió státusz bit	0	Konverzió kész	
			1	Konverzió fut	
Bit 3	CHS0	Analog csatorna választó bitek	000	0. csatorna	
			001	1. csatorna	
			010	2. csatorna	
Bit 4	CHS1		011	3. csatorna	
			100	4. csatorna	
			101	5. csatorna	
Bit 5	CHS2		110	6. csatorna	
			111	7. csatorna	
Bit 6	ADCS0		AD konverter órajel választó bitek	00	FOSC/2
				01	FOSC/8
Bit 7	ADCS1			10	FOSC/32
				11	FRC

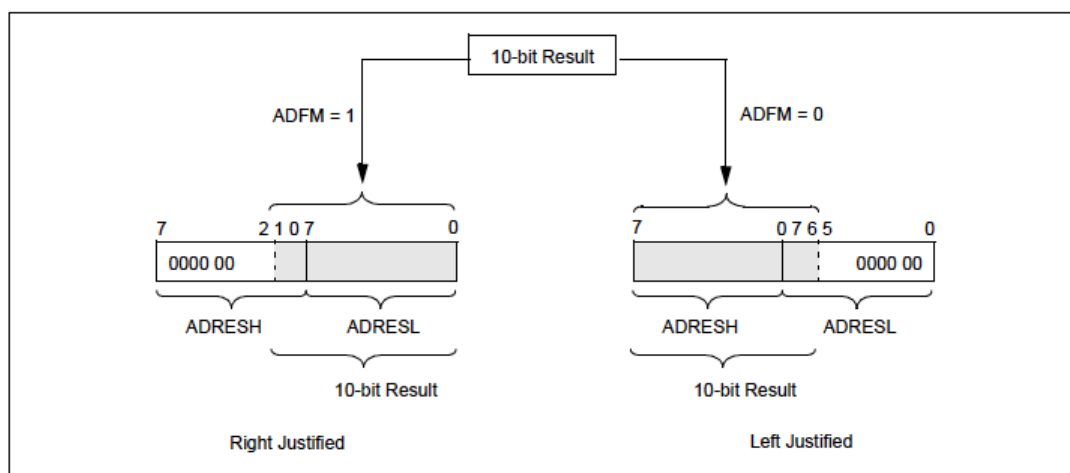
Amint azt az *10. táblázatban* láthatjuk, ez a regiszter használható a modul bekapcsolására, a konverzió indítására, a csatorna és az órajel kiválasztására.

Az ADCON1 regiszter 0-3 bitjei az analóg port konfigurálásra szolgálnak. Ezzel a 4 bittel állíthatjuk be, hogy az analóg port melyik lábát használjuk analóg bemenetként, és melyiket digitális ki/bemenetként. Ezen kívül vannak olyan beállítási lehetőségek is, amelyek használatával az analóg port bizonyos lábait referencia feszültség megadására használhatjuk. Az *11. táblázatban* láthatjuk a beállítási lehetőségeket.

Az A/D konverzió 10 bites eredménye az ADRESH és az ADRESL regiszterekben tárolódik el. Az ADCON1 regiszter 7. ADFM bitje határozza meg, hogy hogyan ábrázoljuk a konverzió eredményét a fent említett két regiszterben. Ha az ADFM értéke 0 akkor a 10 bites eredmény felső 8 bitje az ADRESH regiszterbe kerül, a maradék két bit pedig az ADRESL felső két bitjében tárolódik. Ha az ADFM értéke 1 akkor az eredmény alsó 8 bitje az ADRESL regiszterbe kerül, és a maradék két bit pedig az ADRESH alsó két bitjébe kerül. Az *37. ábra* szemlélteti ezt a két beállítási lehetőséget.

11. táblázat. Analóg port beállítási lehetőségek

PCFG3- PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0
0000	A	A	A	A	A	A	A	A
0001	A	A	A	A	+VREF	A	A	A
0010	D	D	D	A	A	A	A	A
0011	D	D	D	A	+VREF	A	A	A
0100	D	D	D	D	A	D	A	A
0101	D	D	D	D	+VREF	D	A	A
011x	D	D	D	D	D	D	D	D
1000	A	A	A	A	+VREF	-VREF	A	A
1001	D	D	A	A	A	A	A	A
1010	D	D	A	A	+VREF	A	A	A
1011	D	D	A	A	VREF+	VREF-	A	A
1100	D	D	D	A	VREF+	VREF-	A	A
1101	D	D	D	D	VREF+	VREF-	A	A
1110	D	D	D	D	D	D	D	A
1111	D	D	D	D	VREF+	VREF-	D	A



37. ábra. Az A/D konverzió eredményének tárolási módjai (forrás: [7])

Az AD konfigurációs (config) regisztereit után most nézzük meg az inicializáló függvényt, amit a szoftver indításakor egyszer kell lefuttatni. Ahogy az 38. ábrán látjuk, a függvényünk „int” (integer

- egész) típusú, neve `ad_init`, és nincs visszatérési értéke, amire a paraméterlistájában található „void” kifejezés utal. A regiszterek értékeit hexadecimális formátumban adtam meg, az `ADCON0` regiszter tartalma `0b10000000` binárisan ábrázolva. A 10. táblázat alapján tehát láthatjuk, hogy a konverzió órajelét a rendszer órajel 1/32 részére állítottam be. A PIC16F877 adatlapja [7] 20 MHz rendszer órajel frekvenciához az `FOSC/32` beállítást javasolja, ezért választottam én is ezt az opciót. Szintén a 10. táblázatból láthatjuk, hogy a 0. csatornát választottam ki a konverzió forrásaként. Ez viszont csak egy alapértelmezett inicializáló beállítás, ha több AD csatornát is használunk, ezt az értéket a konverzió megkezdése előtt mindig külön módosítani kell. Az `ADCON1` regiszter bináris értéke `0b10000000`, vagyis az `ADFM` bit értéke 1, tehát a jobbra igazított formátumot használom. A `PCFG3-PCFG0` bitek értéke `0000` tehát a teljes portot analóg bemenetként konfiguráltam. Mivel a `PORTA` és `PORTE` lábaira a két analóg bemeneten kívül nem kötöttem semmit, ezért lehet mindegyik láb analóg bemenet. A `TRISA` minden bitjének 1-re állítása a teljes `PORTA`-t bemenetként konfigurálja.

```

24  int ad_init(void)
25  {
26      ADCON0 = 0x80;           //FOSC32, 0. csatorna, AD kikapcsolva
27      ADCON1 = 0x80;           //Jobbra igazított formátum, PORTA analóg
28      TRISA = 0xFF;           //PORTA bemenet
29  }

```

38. ábra. AD inicializáló függvény

```

31  int ref_signal()
32  {
33      unsigned int ad_result;
34      ADCON0 = 0x81;           //AD modul bekapcsolása
35
36      ADCON0bits.GO = 1;       //Konverzió indítása
37      while(ADCON0bits.GO == 1) //Pollingolás
38      {}
39      ad_result = ((ADRESH<<8) + ADRESL) * 2 + 700;
40
41      return ad_result;
42  }

```

39. ábra. Alapjel beolvasása

Az alapjel beolvasására és feldolgozására a `ref_signal` függényt használom. A függvény létrehoz egy „unsigned int” (előjel nélküli egész) típusú változót, melynek neve `ad_result`. Az „int” típus 16 bit adat tárolására alkalmas, ennek „unsigned” változata csak pozitív számok tárolására képes. A 10 bites eredményünk tehát elfér majd ebben a változóban.

Az `ADCON 0` regiszter értékét `0x81`-re állítom, tehát bekapcsolom az AD modult. A

következő sorban is csak 1 bitet állítok át az ADCON0 regiszterben, itt viszont egy másik megoldást használok. A 36. sorban található kifejezés azt jelenti, hogy az ADCON0 regiszterhez bit szinten férhetünk hozzá. A tagoperátor után látható GO annak a bitnek a neve, amihez hozzá kell férni. Ezzel a kifejezéssel tehát elindítom a konverziót.

A 37. sorban látható „while” ciklus pollingolásra szolgál. A pollingolás azt jelenti, hogy folyamatosan figyeljük egy esemény bekövetkezését. Pollingolás közben a program nem tud más feladatot végrehajtani, mivel az üres while ciklus fut le újra és újra, ameddig az AD konverzió nem fejeződik be, és a GO bit értéke 0 nem lesz.

Miután ez az esemény bekövetkezett, a program kikerül a while ciklusból és tovább fut. A 39. sorban történik az eredmény kiértékelése. Mivel jobbra igazított formátumot használunk az ADRESH regiszter alsó két bitjébe tárolódik el a 10 bites eredmény felső két bitje. Az ADRESH regiszter értékét ezért 8 bittel balra kell tolni. Erre szolgál a << operátor. Ezután hozzáadjuk az ADRESL regiszter értékét, amely 10 bites eredmény alsó 8 bitjét tartalmazza. Így kerül tehát bele eltolással a két 8 bites regiszter értéke a 16 bites „int” típusú változóba. A konverzió eredményéből azonban még kiszámolom az alapjel értékét. Az eredményt először 2-vel szorzom, majd hozzáadok 700-t. Az ad_result változó értéke így tehát 700 és 2748 között változhat, 2-es lépésekben, a potenciométer beállításának függvényében.

```
44 int cur_in()
45 {
46     unsigned int ad_result;
47     ADCON0 = 0x89;           //AD modul bekapcsolása, 1-es csatorna
48
49     ADCON0bits.GO = 1;       //Konverzió indítása
50     while(ADCON0bits.GO == 1) //Pollingolás
51     {}
52     ad_result = (ADRESH<<8) + ADRESL;
53
54     return ad_result;
55 }
```

40. ábra. Motor áramának mérése

A cur_in függvény a motor áramának mérésére szolgál. Nagyon hasonlít a ref_signal függvényre, azzal a különbséggel, hogy a 47. sorban az ADCON0 regiszter értékét 0x89-re állítjuk, vagyis átváltunk az 1-es analóg csatornára. Az eredmény feldolgozásánál is csak egyszerűen elhelyezzük eltolással az ADRESH és ADRESL értékét az ad_result változóban. A program nem végez rajta semmi számolást, mivel ennek beállítását csak mérések segítségével lehet elvégezni.

4.3. A fordulatszám mérés

A motor fordulatszámának méréséhez a mikrovezérlő CCP modulját használtam fel. A CCP modul három módban képes működni: capture, compare, PWM. Én ezek közül a méréshez a capture üzemmódot használom. A CCP modul beállítására a CCPxCON regiszter szolgál. Az x helyére 1-t vagy 2-t kell írni, mivel a PIC 16F877 mikrovezérlő kettő darab CCP modullal rendelkezik. Fordulatszám méréshez az egyes modult használom. Az 12. táblázat mutatja be a CCPxCON regiszter felépítését

12. táblázat. CCPxCON regiszter

Bit 0	CCPxM0	CCPx üzemmód választó bitek	0000	CCP modul kikapcsolva
			0100	Capture, minden lefutó élre
Bit 1	CCPxM1		0101	Capture, minden felfutó élre
			0110	Capture, minden 4. Felfutó élre
Bit 2	CCPxM2		0111	Capture, minden 16. Felfutó élre
			1000	Compare, kimenet 1 egyezésnél
			1001	Compare, kimenet 0 egyezésnél
Bit 3	CCPxM3		1010	Compare, megszakítás egyezésnél
			1011	Compare, speciális esemény
			11xx	PWM üzemmód
Bit 4	CCPxY	PWM legalacsonyabb helyiértékű bitjei	capture	Nem használt
			compare	Nem használt
Bit 5	CCPxX		PWM	A kitöltési tényező alsó két bitje
Bit 6	-			
Bit 7	-			

Capture üzemmódban az RC2/CCP1 lábon bekövetkező esemény hatására a timer1 16 bites időzítő/számláló értéke átmásolódik a CCPR1H és CCPR1L regiszterekbe. A capture üzemmód segítségével lehetőségünk nyílik két esemény között eltelt idő nagyon pontos mérésére. Ehhez

azonban előbb szükségünk lesz a timer1 felkonfigurálására. Erre szolgál a T1CON regiszter, amelyet az 13. táblázat mutat be.

13. táblázat. T1CON regiszter

Bit 0	TMR1ON	Timer bekapcsolása	0	Timer 1 megáll
			1	Timer 1 fut
Bit 1	TMR1CS	Órajel forrás választó bit	0	Belső órajel
			1	Külső órajel
Bit 2	<u>T1SYNC</u>	Szinkronizáció kontrol bit	0	Külső órajelhez szinkronizál
			1	Nem szinkronizál
Bit 3	T1OSCEN	Oscillátor engedélyező bit	0	Oscillátor kikapcsolva
			1	Oscillátor bekapcsolva
Bit 4	T1CKPS0	Órajel előosztó bitek	11	1/8 előosztó
			10	1/4 előosztó
Bit 5	T1CKPS1		01	1/2 előosztó
			00	1/1 előosztó
Bit 6	-			
Bit 7	-			

A fordulatszám mérés inicializálásához a rpm_init függvényt használom, amely implementálása a 41. ábrán látható. A 28. sorban a T1CON értékét 0x30-ra állítom, amely az x. táblázat alapján azt jelenti, hogy 1/8 előosztást állítottam be, és az órajel forrásként a processzor belső órajelét választottam. A PIC 16F877 adatlapjából kiolvasható, hogy belső órajel használata esetén a timer1 léptetési frekvenciája $f_{osc}/4$, vagyis a rendszer órajel negyede. A timer1 léptetési idejét a következő képlettel határozhatjuk meg:

$$t_{step} = \frac{1}{\frac{f_{osc}}{4 \cdot prescale}} = \frac{1}{\frac{20000000}{4 \cdot 8}} = 1.6 \cdot 10^{-6} s = 1.6 \mu s \quad (4)$$

ahol Tstep a lépésidő, és Fosc a belső órajel.

20MHz frekvenciájú kristály oszcillátor használata mellett, a beállított 1/8 előosztóval a timer1

léptetési ideje 1.6 us. Ez később fontos lesz a fordulatszám számításánál.

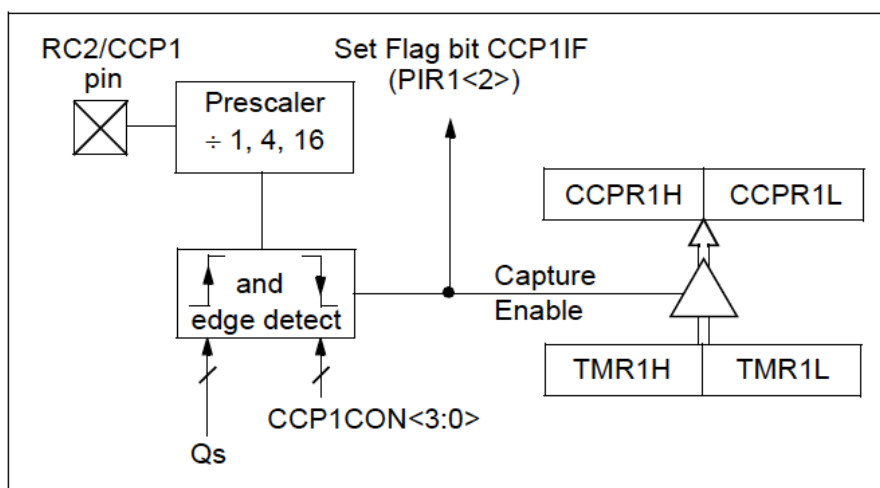
```

25 int rpm_init(void)
26 {
27     CCP1CON = 0x00;           //CCP modul kikapcsolva
28     T1CON = 0x30;            //Timer kikapcsolva, 1/8 előosztó, belső órajel
29     INTCON = 0x00;           //Megszakítások tiltása
30     TRISCbits.TRISC2 = 1;    //PORTC RC2/CCP1 láb bemenet
31     PIE1 = 0x00;             //Perifériális megszakítások tiltása
32     PIR1 = 0x00;             //Megszakítás jelzőbitek törlése
33
34     TMR1H = 0x00;            //Timer1 felső bájt törlése
35     TMR1L = 0x00;            //Timer1 alsó bájt törlése
36
37     T1CONbits.TMR1ON = 1;    //Timer1 léptetésének indítása
38     CCP1CON = 0x05;          //Capture mód, capture minden felfutó éle
39 }

```

41. ábra. Fordulatszám mérés inicializálása

A 29. és a 31. sorokban a megszakításokat letiltottam. Ahogyan azt a 42. ábrán láthatjuk, capture esemény hatására a PIR1 regiszter CCP1IF interrupt jelző bitje 1-es értéket vesz fel. A capture események detektálását és kezelését végezhetném egy interrupt segítségével is, viszont mivel a mikrovezérlőnek van ideje csak a fordulatszám méréssel foglalkozni, ezért inkább az egyszerűbb CCP1IF pollingolás módszerét választottam. Ez az oka annak, hogy a megszakítások tiltásra kerültek. A pollingoláshoz szükség lesz az interrupt jelzőbit szoftverből történő törlésére, ezért az inicializálás során egyből kinullázom a PIR1 interrupt flageket tartalmazó regiszter értékét. A capture üzemmód használatához az RC2/CCP1 lábat bemenetként kell konfigurálni, ezt a 30. sorban végzem el.



42. ábra. A capture üzemmód blokk diagrammja

A 34. és 35. Sorban kinullázom a timer1 léptető regisztereinek értékét, majd a következő sorban elindítom a timer1 léptetését, és végül a CCP1CON 0x05 értékadásával elindítom a capture

üzemmódot minden felfutó élre.

```
41 int rpm_mes(void)
42 {
43     unsigned long mes_time = 0;
44     unsigned int mes_cnt = 0;
45     unsigned int mes_cnt1 = 0;
46     unsigned int mes_cnt2 = 0;
47
48     TMR1H = 0x00; //Timer1 felső bájt nullázása
49     TMR1L = 0x00; //Timer1 alsó bájt nullázása
50
51     while(!PIR1bits.CCP1IF) //CCP interrupt flag figyelés
52     {
53     }
54     mes_cnt1 = (CCPR1H << 8) + CCPR1L; //Értékek összeadása
55     PIR1bits.CCP1IF = 0; //IF nullázása
56
57     while(!PIR1bits.CCP1IF) //CCP interrupt flag figyelés
58     {
59     }
60     mes_cnt2 = (CCPR1H << 8) + CCPR1L; //Értékek összeadása
61     PIR1bits.CCP1IF = 0; //IF nullázása
62
63     if(mes_cnt2 > mes_cnt1) //Értékek összehasonlítása
64     {
65         mes_cnt = mes_cnt2 - mes_cnt1;
66     }
67     else if(mes_cnt1 > mes_cnt2)
68     {
69         mes_cnt = (65535 - mes_cnt1) + mes_cnt2;
70     }
71
72     mes_time = 37500000 / mes_cnt; //Fordulatszám kiszámítása
73
74     return mes_time;
75 }
```

43. ábra. Fordulatszám mérő programrészlet.

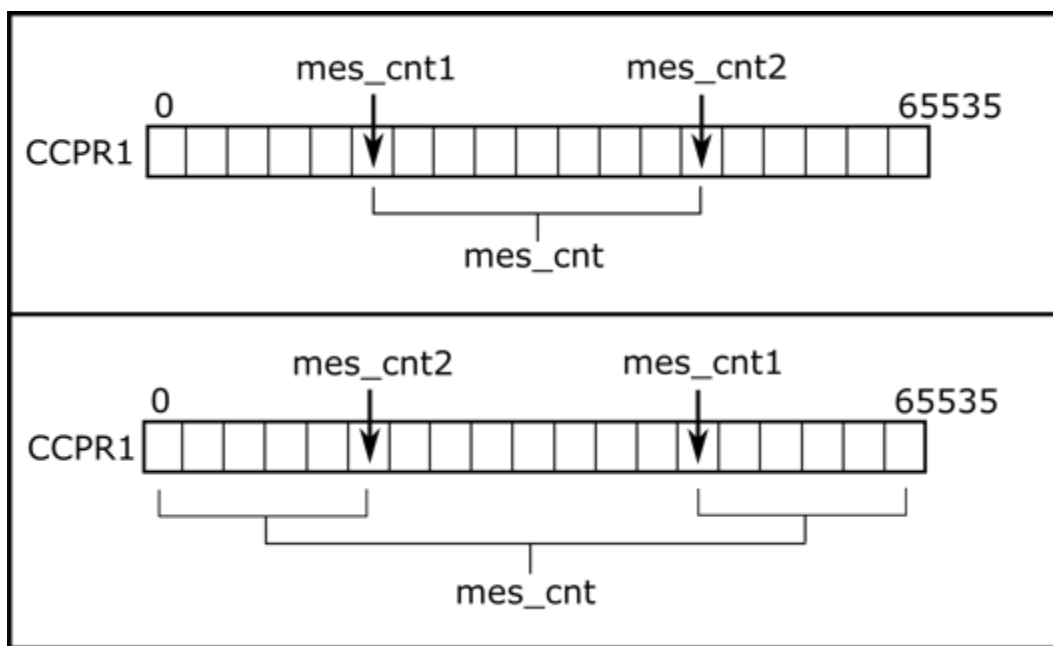
Az rpm_mes függvény meghívásával mérem a motor fordulatszámát. Ennek a függvénynek a visszatérési értéke a tényleges motor fordulat. A méréshez szükség van négy változóra, melyek közül három „unsigned int” típusú. Ezekben tárolom a timer1 elkapott értékeit, valamint a harmadikban a két capture esemény között eltelt lépések számát. A negyedik változó „unsigned long” típusú, amely 32 biten képes egész számokat tárolni előjel nélkül. Erre azért van szükség, mert ebbe a változóba kerül egy számolás útján a fordulatszám érték. Ahogy a 43. ábrán a 72. sorban láthatjuk, a kalkulációban szerepel a 37500000 szám, amely nem férne bele egy 16 bites „unsigned int” típusú változóba, ami csak 0 és 65535 között képes értékeket tárolni.

A változók deklarálása, és a timer1 léptető regisztereinek nullázása után a program pollingolással elkezdni figyelni, hogy mikor következik be egy capture esemény, vagyis mikor érkezik egy felfutó él az RC2/CCP1 lábra. Ha ez megtörténik, a CCP1IF bit értéke 1-re vált, és a program elmenti a CCPR1H és CCPR1L értékét a mes_cnt1 változóba. Valójában a timer1 capture

eseménykor aktuális értéke kerül mentésre. A két darab 8 bites regiszter értékét az AD átalakításnál már látott eltolásos módszerrel mentem a 16 bites változóba.

Az interrupt jelzőbit szoftveres törlése után a program vár a következő capture eseményre. Ekkor már a mes_cnt2 változóba mentjük a CCPR1H és CCPR1L aktuális értékét.

Ezek után a mes_cnt1 és mes_cnt2 változóban szerepel két érték, amelyek különbsége megadja, hogy a két capture esemény között (vagyis egy fordulat megtétele alatt) a timer1 mennyi lépést tett meg. Ideális esetben a mes_cnt2 értéknek nagyobbnak kellene lennie a mes_cnt1 értékénél, ahogyan ezt az 44. ábra felső részén láthatjuk. Viszont alacsony fordulatszámok esetén, ahol két capture esemény között hosszabb idő telik el, előfordulhat, hogy a két capture esemény között a timer1 értéke eléri a 65535-öt, majd újakezdi a számlálást 0-tól. Ezt az esetet láthatjuk az 44. ábra alsó felében. A pontos fordulatszám méréshez ezeket az eseteket azonosítani és kezelni kell, szoftveres úton.



44. ábra. A két mentett érték elhelyezkedésének módjai

Az 43. ábra (fordulatszám programrész) 36-70 sorai között található egy feltételes szerkezet, amellyel megvizsgálom, hogy melyik mentett érték a nagyobb. Ha a 44. ábrán az első eset áll fent, vagyis a második mentett érték a nagyobb, akkor az eltelt idő számítására szolgáló képlet:

$$\text{mes_cnt} = \text{mes_cnt2} - \text{mes_cnt1} \quad (5)$$

ahol mes_cnt a két capture esemény között eltelt lépések száma, mes_cnt2 a timer1 értéke a második capture esemény időpillanatában, és mes_cnt1 a timer1 értéke az első capture esemény

időpillanatában.

Ha viszont a második eset áll fent, és történt egy túlsordulás, akkor először meg kell nézni, hogy az első mentett érték és a túlsordulás között mennyi idő telt el, majd ehhez hozzá kell adni a túlsordulás és a második capture esemény között eltelt időt. A képlet ebben az esetben tehát:

$$mes_cnt = (65535 - mes_cnt1) + mes_cnt2. \quad (6)$$

A 72. sorban történik meg a fordulatszám értékének meghatározása. A képlet a programban egyszerűsítve szerepel. A mes_cnt változóban tárolt érték a két capture esemény között eltelt timer1 lépések száma. Ahhoz, hogy ebből periódus időt kapjunk, meg kell szorozni a timer1 lépés idejével. Ezt az értéket korábban az inicializálásnál már kiszámoltam, az eredmény $1.6\mu s$ lett. A fordulatszám meghatározásához a következő képletet használtam:

$$mes_{time} = \frac{60 s}{mes_{cnt} * 1.6 \mu s} \quad (7)$$

ahol mes_time a fordulatszám értéke, és mes_cnt a két capture esemény között eltelt lépések száma. Ha közös mértékegységeket használunk, a képlet a következő alakra egyszerűsíthető:

$$mes_{time} = \frac{60000000 \mu s}{mes_{cnt} * 1.6 \mu s} = \frac{37500000}{mes_{cnt}} \quad (8)$$

Belátható azonban az is, hogy a timer1 túlsordulása miatt ennek a fordulatszám mérési megoldásnak van egy alsó határa, mivel nem tudunk 65535 lépésnél többet megszámolni. A legkisebb mérhető fordulatszám:

$$min_{rpm} = \frac{37500000}{65535} = 572.21 \quad (9)$$

ahol min_rpm a legkiseb mérhető fordulatszám. Természetesen, ha van a mérésnek alsó korlátja akkor van felső is, ez pedig az egy lépéshez tartozó fordulatszám, ami:

$$max_{rpm} = \frac{37500000}{1} = 37500000 \quad (10)$$

ahol max_rpm a legnagyobb mérhető fordulatszám. A felső határ elég magas, így nem kell tartani

attól, hogy ezt az értéket elérnénk. Mivel az alapjel előállításánál meghatároztam, hogy a legkisebb beállítható fordulatszám 700 rpm lesz, így nem okoz problémát az 572-es legkisebb mérhető fordulat. Ha viszont a későbbiekben szeretnénk kisebb fordulatszámokat is mérni az eszközzel, több lehetőség is adódik a mérési tartomány eltolására, vagy kibővítésére.

A mérés tartománya kibővíthető, ha megszakítások alkalmazásával számoljuk, hogy két capture esemény között a timer1 hányszor csordult túl, és a kapott értéket figyelembe vesszük a fordulatszám számításakor. Ezzel a megoldással gyakorlatilag közel végtelenül kicsi fordulatszámok mérése is lehetséges úgy, hogy a maximálisan megmérhető fordulatszám értéke nem csökken.

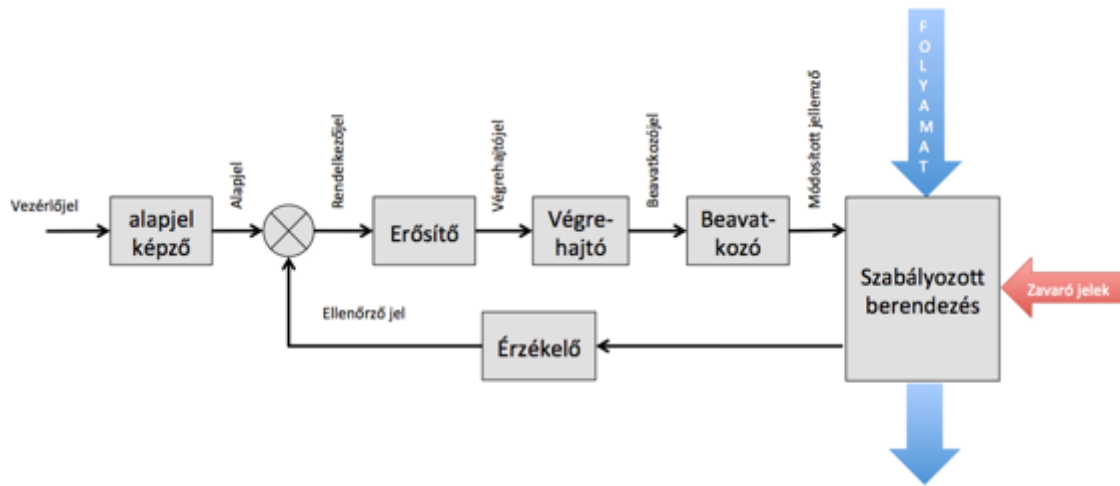
A mérési tartomány eltolására két módszer is létezik. Az egyik a rendszer órajel csökkentése, mellyel gyakorlatilag a timer léptetési sebességét csökkentjük. A másik megoldás, hogy a fordulatszám mérésére használt tárcsára nemcsak egy, hanem több furatot is elhelyezünk. Így a két capture esemény között eltelt időt csökkentjük. Természetesen mindegyik megoldásnál módosítani kell a fordulatszám meghatározására szolgáló képletet.

Ezek után belátható, hogy a capture modul felhasználásával nagyon pontos, gyors, és jól konfigurálható fordulatszám mérő programot hozhatunk létre.

4.4. A szabályozás

Az egész projekt egyik legfontosabb része a szabályozó algoritmust futtató programrész. Ezért mielőtt rátérnék erre, szeretném részletesebben ismertetni a szabályozás fogalmát, valamint a különböző szabályozókat.

A szabályozás az irányítástechnika tudományág egyik része. Az irányítástechnikában két fajta irányítási módot különböztetünk meg. Ezek a vezérlés és a szabályozás. A két irányítás közötti legfőbb különbség, hogy míg a vezérlés hatáslánca nyitott, addig a szabályozásé zárt. A szabályozás során az irányított jellemző valós és előírt értékeinek különbsége alapján állítjuk elő a beavatkozó jelet, amely hatására a különbség csökkenni fog. A szabályozás hatásláncát szabályozási körnek nevezzük amelyet a 45. ábrán láthatunk.



45. ábra. A szabályozási kör

Vegyük most végig a szabályozási kör elemeit, és feladataikat:

- Alapjel képző: Feladata, hogy a vezérlő jelből különbség képzésre alkalmas jelet állítson elő (potenciométer).
- Különbségképző: Az alapjel és a visszacsatolt ellenőrző jel különbségével arányos jelet állít elő.
- Erősítő: A rendelkező jelet erősíti a végrehajtó számára feldolgozható mértékre.
- Végrehajtó: A végrehajtó jel alapján előállítja a beavatkozáshoz szükséges jelet (CCP modul PWM üzemmódban).
- Beavatkozó: Befolyásolja a módosított jellemzőt (PWM jel).
- Szabályozott berendezés: Egy folyamatban működő önálló eszköz, aminek valamilyen jellemzőjét szabályozni szeretnénk.
- Érzékelő: A szabályozott jellemző értékét érzékeli, és ezzel arányos ellenőrző jelet állít elő.

A szabályozási kör jelei:

- Vezérlőjel: Az alapjel képző bemenő jele (ha az alapjel képző egy potenciométer, akkor a kar pozíciója).
- Alapjel: A szabályozott jellemző kívánt értékével arányos különbség képzésére alkalmas jel.
- Ellenőrző jel: A szabályozott jellemző aktuális értékével arányos különbség képzésére alkalmas jel.
- Rendelkező jel: Más néven hibajel, a szabályozott jellemző kívánt és aktuális értékének különbségével arányos jel.
- Végrehajtó jel: A felerősített rendelkező jel, amely a végrehajtó szerv bemenő jele.

- Beavatkozó jel: a beavatkozó szerv bemeneti jele.
- Zavaró jelek: Azok a külső hatások, amelyek a szabályozott berendezésre hatnak és a szabályozott jellemzőnk értékét befolyásolják.

A szabályozásokat két csoportba sorolhatjuk az alapjel szerint. Értéktartó szabályozásról akkor beszélünk, ha az alapjel értéke állandó, és a szabályozott berendezésre ható külső zavaró jelek változnak időben. Követő szabályozásnak nevezzük azt, amikor az alapjel értékét időben folyamatosan változtatjuk, és a szabályozott jellemzőnek ezt a változást minél gyorsabban le kell követnie. Az általam megvalósított motor szabályozás mindkét csoportba beletartozik, hiszen a motornak követnie kell az alapjel változását is, azonban ha az alapjelet fixen tartjuk és a motor tengelyén növeljük a terhelést, az ebből adódó fordulatszám változást is korrigálnia kell a szabályozásnak.

4.5. A PID szabályozó

Napjainkban a PID egy nagyon elterjedt szabályozási módszernek számít, gyakran használják ipari szabályozók megvalósításához, mivel szoftveresen könnyen implementálható, megbízható és gyors. A PID név a Proportional Integral Differential szavak rövidítése, amely arra utal, hogy a szabályozó a végrehajtó jel számításakor figyelembe veszi a jelenlegi hibát (arányos tag), a múltbéli hibák átlagát (integráló tag), és a jövőbeli hibák várható értékét (differenciáló tag).

A PID szabályozó matematikai leírása a következő:

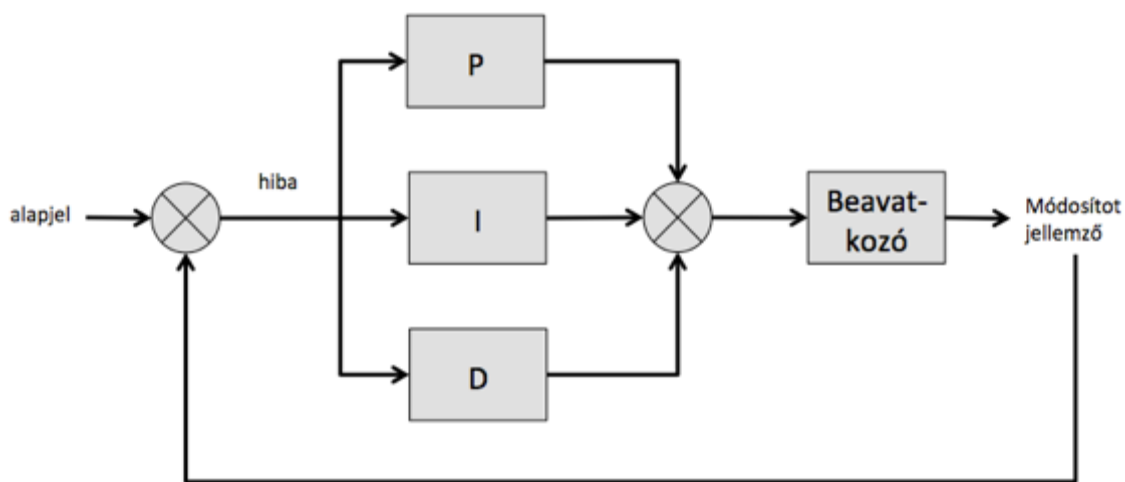
$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \frac{d}{dt} \cdot e(t) \quad (11)$$

ahol: $u(t)$ a szabályozó kimenete, vagyis a beavatkozó jel, $e(t)$ a szabályozó bemenete, vagyis a hibajel, K_p az arányos tagot súlyozó konstans, K_i az integráló tagot súlyozó konstans, és K_d a differenciáló tagot súlyozó konstans. A fenti képletet folytonos idejű szabályozás esetén használhatjuk. Diszkrét idejű szabályozásnál, mint az általam alkalmazott digitális szabályozás, a fenti képletet az alábbi formára kell átalakítani:

$$u(n) = K_p \cdot e(n) + K_i \cdot \sum_{k=0}^n e(k) + K_d \cdot (e(n) - e(n-1)) \quad (12)$$

Ahol $u(n)$ a szabályozó kimenete diszkrét idő esetén, n az aktuális diszkrét idő lépés értéke, és $e(n)$ a hiba értéke. Az átalakításokra azért volt szükség, hogy a képletet le tudjuk programozni, ugyanis a számítógépek és a mikrovezérlők nem tudják elvégezni az integrálás és a deriválás műveleteket, ezért ezeket közelítő módszerekkel kell helyettesíteni.

A fenti két képlet alapján belátható, hogy a PID szabályozó a három alap szabályozó matematikai leírásának összegzéséből áll össze. Ezek alapján tehát az 46. ábra a PID szabályozó egyszerűsített hatásláncát mutatja.



46. ábra. PID szabályozó

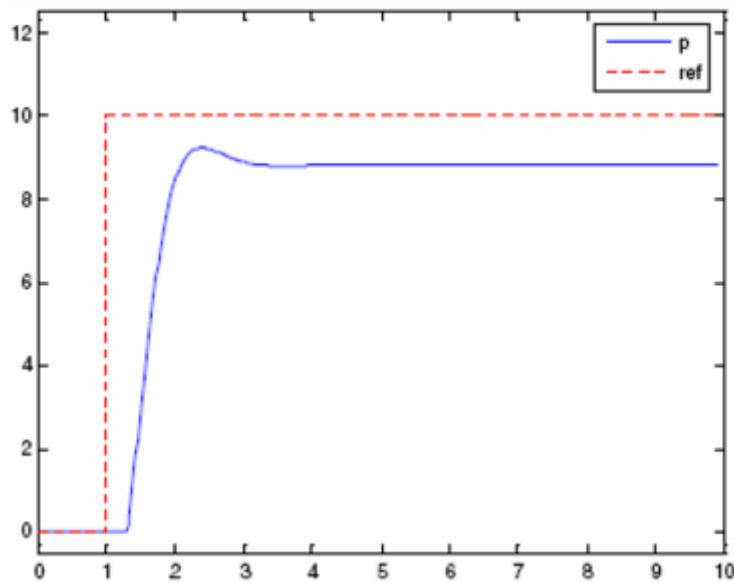
4.5.1. P szabályozó tag

A P tag matematikai leírása diszkrét idejű szabályozás esetén a következő:

$$P = K_p \cdot e(n) \quad (13)$$

Vagyis a P tag az aktuális hibajellel arányos beavatkozó jelet szolgáltat, ahol a K_p konstans nevezhetjük az arányos tag erősítésének.

Az 47. ábra alapján láthatjuk, hogy a P tag a bemenetére adott egységugrás jelet a kimenet csak bizonyos nagyságú hibával képes elérni. Ez alapján megállapíthatjuk, hogy a P tag önmagában nem elegendő arra, hogy a szabályozással szemben támasztott minőségi igényeinket kielégítse. A K_p erősítés növelésével ez a hiba csökkenthető, viszont egy szinten túli növelése már a rendszer instabilitását okozhatja.



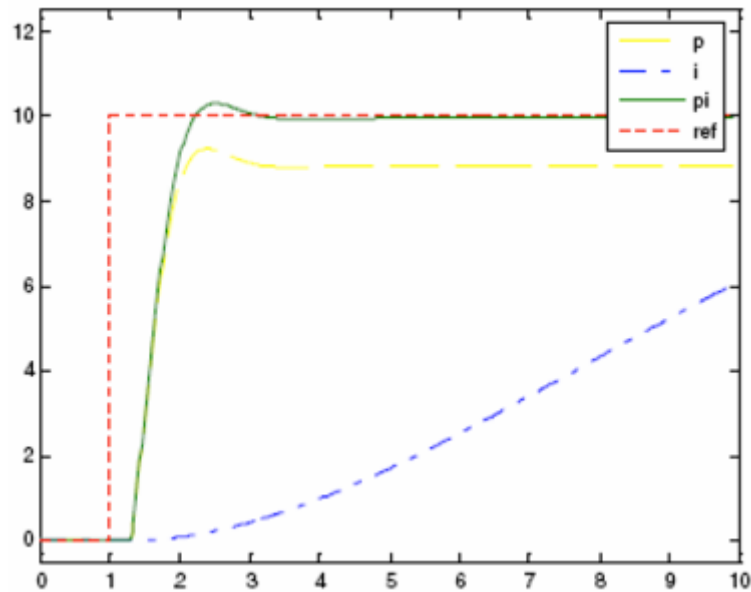
47. ábra. Az arányos tag ugrásválasza (forrás: [12])

4.5.2. Az I szabályozó tag

Az I tag matematikai leírása diszkrét idejű szabályozás esetén a következő:

$$I = K_i \cdot \sum_{k=0}^n e(k) \quad (14)$$

A képlet alapján mondhatjuk, hogy az I tag olyan beavatkozó jelet állít elő, amely a hiba korábbi értékeinek összegével arányos. Ezen tulajdonságának köszönhetően az I tag képes megszüntetni azt a hibát, amit a P tag egy szinten túl már nem volt képes. A 48. ábrán láthatjuk a P az I és a PI tagok ugrásválaszát. Az I tag önmagában való használata nagyon lassú szabályozást eredményezne. A PI tagok összegzése azonban már egy jól használható szabályozást valósít meg.



48. ábra. A PI tag ugrásválasza (forrás: [12])

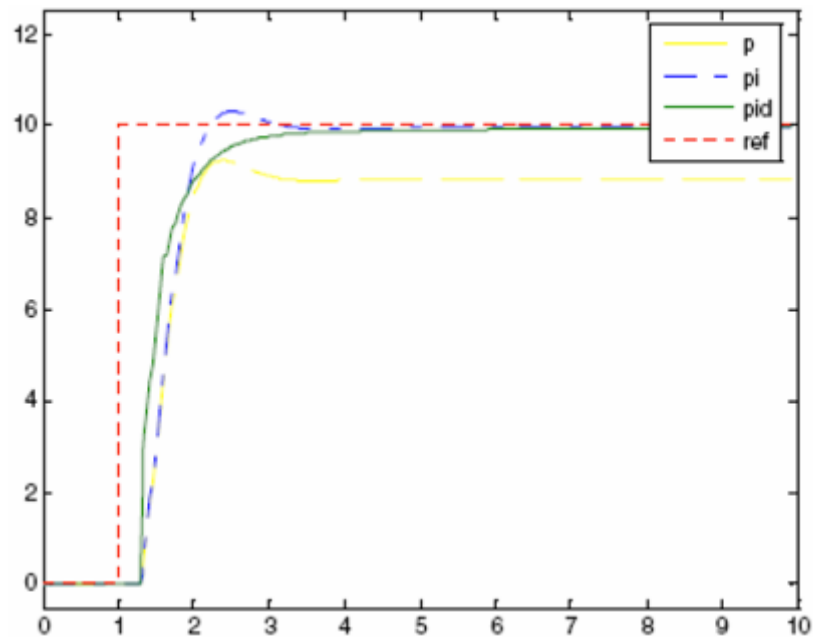
4.5.3. A D szabályozó tag

A D tag matematikai leírása diszkrét idejű szabályozás esetén a következő:

$$D = K_d \cdot (e(n) - e(n - 1)) \quad (15)$$

Sajnos számítógéppel nem tudunk deriválást végrehajtani, ezért szükséges a fent látható közelítés alkalmazása. A képlet alapján könnyű belátni, hogy a D tag olyan beavatkozó jelet szolgáltat, amely a hiba változási sebességével, vagyis meredekségével arányos. D tag használatával gyorsíthatjuk a szabályozásunkat, mivel differenciáló tag az alapjel gyors változásaira nagy beavatkozó jelet szolgáltat. A D tag önmagában nem lenne alkalmas a szabályozás megvalósítására, azért mindig PD vagy PID formában használjuk. Ha a „ K_d ” konstans túl nagynak választjuk meg, akkor a rendszerünk instabillá válhat, és a zavarjelekre is sokkal érzékenyebbé válik.

A 49. ábrán láthatjuk, hogy a D tag használatával meredekebb, felfutó élet kapunk, valamint a P és D tagok „túllövését” is megszüntethetjük. A három tag együttes alkalmazásával egy gyors és pontos szabályozót kapunk, amely hangolása elvégezhető a három erősítési konstans beállításával.



49. ábra. A PID tag ugrásválasza (forrás: [12])

4.6. PID szabályozó szoftveres implementálása

A korábban bemutatott PID szabályozóra vonatkozó képletek alapján könnyedén megírható egy ilyen szabályozást megvalósítható mikrovezérlő program. Az 50. ábrán látható az általam használt PID szabályozó programrészlet.

A függvény előtt implementált globális változók és a konstansok definiálása valójában a main függvény elején történik, és a main függvényen belül történik a pid_control függvény meghívása is. A függvény hívása után létrejön az „error” változó, amely értékül az alapjel és a mért fordulatszám különbségét kapja. Ezután történik a P, I, D tagok értékeinek kiszámítása a korábban ismertetett képletek alapján. Annak elkerülése érdekében, hogy a szabályozó tagok túlvezéreljék a rendszert, limitálni kell a maximális és minimális értéküket. Erre szolgálnak az egyes tagok számítása után álló feltételes szerkezetek. Ezt a limitálást a tagok összegéből képzett beavatkozó jellel is el kell végezni. A függvény végén az aktuális hiba értéket át kell másolni a prev_error globális változóba, hogy a függvény következő hívásakor rendelkezünk az előző hiba értékével. A függvény végül visszaadja számított U értéket, amelyet a továbbiakban a program a PWM-jel beállításához használ fel.

```

1  #define K_P 100          //P tag er?sítése
2  #define K_I 10;         //I tag er?sítése
3  #define K_D 10;         //D tag er?sítése
4
5  int prev_error, P=0, I=0, D=0, U=0;    //Globális változók
6
7  int pid_control(int PV, int SP)
8  {
9      int error;                //Hiba érték, lokális változó
10
11      error = SP - PV;          //Aktuális hiba számítása
12
13      P = error * K_P;          //P tag számítása
14      if(P > 1000)              //P tag értékének limitálása
15          P = 1000;
16      else if (P < -1000)
17          P = -1000;
18
19      I = I + error * K_I;      //I tag számítása
20      if(I > 1000)              //I tag értékének limitálása
21          I = 1000;
22      else if(I < -1000)
23          I = -1000;
24
25      D = (error - prev_error) * K_D; //D tag számítása
26      if(D > 1000)              //D tag értékének limitálása
27          D = 1000;
28      else if(D < -1000)
29          D = -1000;
30
31      U = P + I + D;            //U beavatkozó jel számítása
32      if(U > 1000)              //U értékének limitálása
33          U = 1000;
34      else if(U < -1000)
35          U = -1000;
36
37      prev_error = error;       //Elz? hiba érték felülírása
38
39      return U;                // visszatérés U értékével
40  }

```

50. ábra. PID szabályozó programrészlet

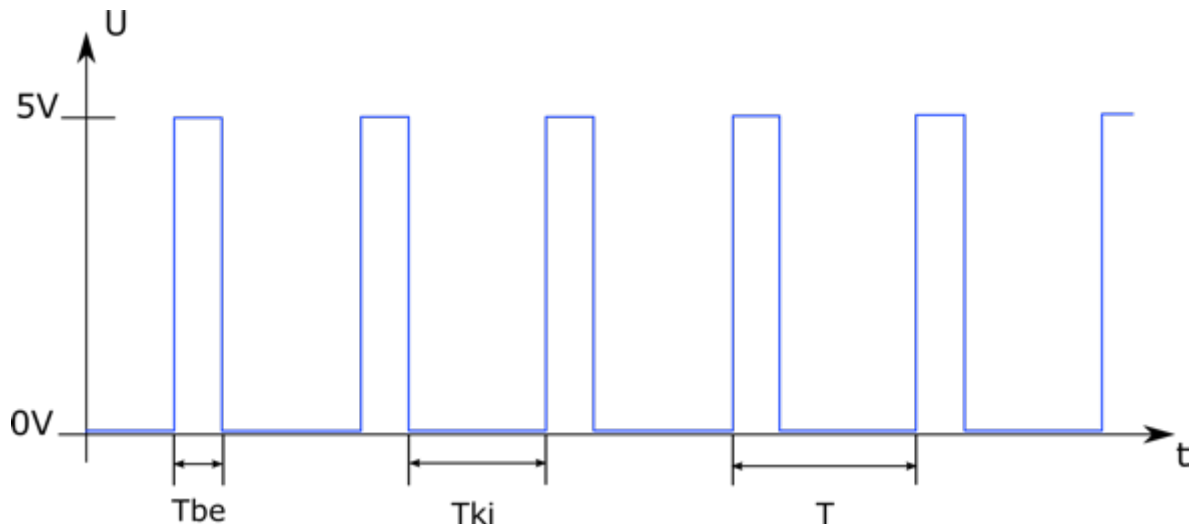
Az 50. ábrán látható konstans értékek, valamint a minimális és maximális szabályozó értékek csak megközelítőleg lettek beállítva, ökölszabályok alapján. Ezen értékek pontos beállítása empirikus úton történik az eszköz tesztelése során.

4.7.A PWM jel előállítása

A PWM jelentése Pulse Widht Modulation, magyarul impulzus szélesség moduláció. Széles körben alkalmazott D/A átalakítási mód. Gyakran használják digitális fényerőszabályozás megvalósítására, de motorok fordulatszámának szabályozására is tökéletesen alkalmazható.

Köztudott, hogy a mikrovezérlők nem képesek analóg feszültség jelet szolgáltatni a kimenetükön. A PWM modul úgy állít elő kvázi analóg kimenő jelet, hogy egy megadott periódus

idejű és megadott kitöltési tényezőjű négyszögjelet ad ki folyamatosan a kimeneten. Ez egy diszkrét feszültség jel, mivel értéke 0V vagy 5V. Viszont ha ezt a négyszögjelet egy DC vezérlésű fogyasztóra kötjük, akkor a fogyasztó idő tartományban a négyszögjel idő szerinti integrálját fogja érzékelni. Az 51. ábrán egy 25%-os kitöltési tényezőjű PWM jele időfüggvényét láthatjuk. Ha ezen az ábrán a feszültséget integráljuk az x és $x+T$ közötti intervallumon, a kapott eredmény 1.25 lesz. A négyszögjel frekvenciáját minden esetben az általa működtető fogyasztóhoz kell beállítani.



51. ábra. PWM jel feszültség/idő diagramja

4.7.1. PWM jel előállítása a CCP modul használatával

A PIC mikrovezérlők CCP modulja (Capture, Compare, PWM) képes a PWM jel előállítására. A PWM üzemmódot a 4.4 (fordulatszám mérése) fejezetben bemutatott CCPxCON regiszter segítségével tudjuk aktiválni. A PIC 16F877 két darab CCP modullal rendelkezik. Én az elsőt használtam a fordulatszám mérésére, így a PWM jel előállítására a másodikat kell használnom. Az üzemmód aktiválásán kívül a CCP2CON regisztert kell majd használnom a 10 bites kitöltési tényező alsó 2 bitjének tárolására. Ez a két bit a regiszter CCP2X és CCP2Y nevű bitjeibe kerül majd be. A PWM használatához szükségünk lesz még a timer2 modulra is.

A timer2 egy 8 bites időzítő, amely léptető órajele a rendszer órajel negyede ($f_{osc}/4$). Ez az időzítő modul rendelkezik 2 bites előosztóval és 4 bites utóosztóval is. A modul konfigurálását a T2CON regiszterrel végezhetjük, amely felépítését az x. táblázatban láthatjuk.

14. táblázat. A T2CON regiszter felépítése

Bit 0	T2CKPS0	Előosztó bitek	00	1-es előosztó
			01	4-es előosztó
Bit 1	T2CKPS1		1x	16-os előosztó
Bit 2	TMR2ON	Timer bekapcsoló bit	0	Timer2 kikapcsolva
			1	Timer2 bekapcsolva
Bit 3	TOUTPS0	Utóosztó bitek	0000	1:1 utóosztó
			0001	1:2 utóosztó
Bit 4	TOUTPS1		0010	1:3 utóosztó
			0011	1:4 utóosztó
Bit 5	TOUTPS2		.	.
			.	.
Bit 6	TOUTPS3		.	.
			1111	1:16 utóosztó
Bit 7	-			

A timer2-höz tartozik egy 8 bites PR2 regiszter is. Bekapcsolt állapotban az órajel frekvenciájával növeli a vezérlő a TMR2 regiszter értékét. Ha a TMR2 eléri a PR2 felhasználó által beállított értékét a TMR2 regiszter értéke 0-ra áll vissza és a TMR2IF megszakítás jelző bit értéke 1 lesz.

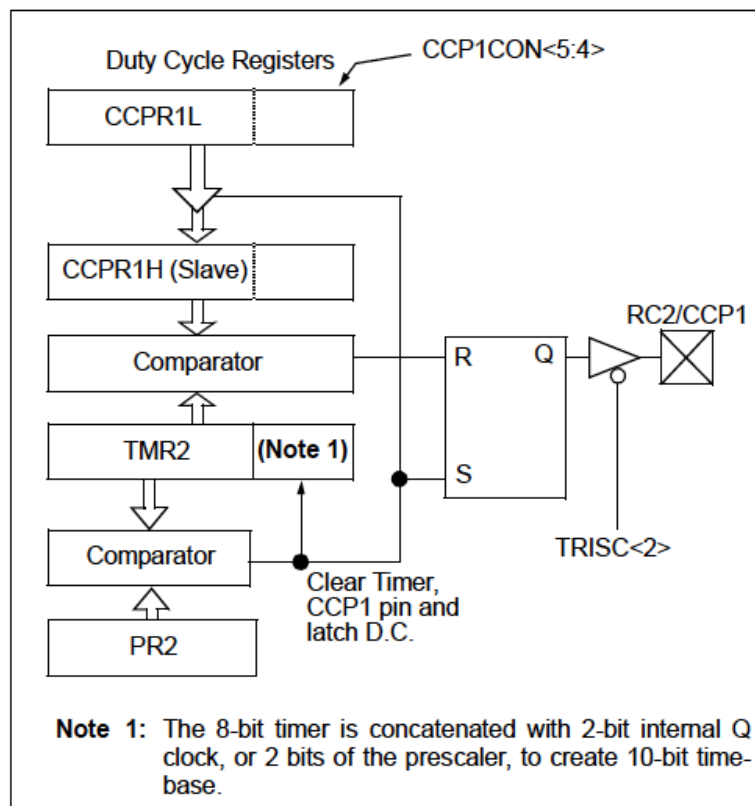
A PWM jel periódus idejét a PR2 regiszter segítségével állíthatjuk be. A periódusidő számítására szolgáló képlet:

$$T_{PWM} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (TMR2 \text{ előosztó}) \quad (16)$$

ahol: T_{pwm} : a PWM jel periódusideje, (PR2): a PR2 regiszter értéke, T_{osc} : a léptető órajel periódusideje, (TMR2 előosztó): a timer2 előosztó értéke. A PWM jel kitöltési tényezőjét a CCPR2L regiszter és a CCP2CON regiszter CCP2X és CCP2Y bitjei segítségével állíthatjuk be. A kitöltési tényezőt a következő képlettel számíthatjuk:

$$PWM_{duty} = (CCPR2L:CCP2CON < 5:4 >) \cdot T_{OSC} \cdot (TMR2 \text{ előosztó}) \quad (17)$$

ahol: PWMduty: a PWM kitöltési tényező, (CCPR2L:CCP2CON<5:4>): a 10 bitesre bővített tároló regiszter, Tosc: a léptető órajel periódusideje, (TMR2 előosztó): a timer2 előosztó értéke.



52. ábra. A PWM modul blokkvázlata (forrás: [7])

Az 52. ábrán láthatjuk a PWM modul egyszerűsített blokkvázlatát. Ha mindent megfelelően konfiguráltunk, és elindítottuk a timer2 léptetését, akkor a TMR2 értéke elkezd növekedni. Ha TMR2 és PR2 egyenlő lesz, akkor a TMR2 regiszter értéke nullázódik, CCPR2L értéke átkerül a CCPR2H regiszterbe, és RC1/CCP2 értéke 1 lesz, ha a PWM kitöltési tényező értéke nem nulla. Ezután a TMR2 értéke ismét növekedni kezd nullától, és ha eléri a CCPR1H értékét az RC1/CCP2 láb nullára vált.

4.7.2. A PWM modul konfigurálása

15. táblázat. PWM frekvenciák és felbontásuk 20MHz órajel esetén

PWM frekvencia	1.22 KHz	4.88 KHz	19.53 KHz	78.12 KHz	156.3 KHz	208.3 KHz
Előosztó értéke	16	4	1	1	1	1
PR2 érték	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximális felbontás	10 bit	10 bit	10 bit	8 bit	7 bit	5.5 bit

A PIC 16F877 mikrovezérlő adatlapja tartalmaz egy táblázatot, amelyben felsorol néhány PWM frekvencia értéket és a hozzájuk tartozó timer2 előosztó és PR2 értékeket, valamint az elérhető maximális felbontást. Ez az *15. táblázat*. DC motorok PWM jellel történő szabályozásához elterjedten a 20KHz-es PWM frekvenciát szokás használni. Nekem tehát a táblázatban szereplő 19.53 KHz-es frekvencia megfelelő, ezért a hozzá tartozó adatokat használom a PWM modul konfigurálásához. Ezeket az értékeket képletekkel is kiszámíthatjuk.

A processzorom órajele 20 MHz, és szeretném a kitöltési tényező felbontását maximálisan kihasználni. A kiinduló adataim tehát:

- Fosc: 20MHz
- N: 10bit

$$Z = 2^N = 2^{10} = 1024 \quad (18)$$

ahol Z a lépés szám.

$$F_{PWM} = \frac{F_{OSC}}{Z} = \frac{20MHz}{1024} = 19.531 KHz \quad (19)$$

ahol Fpwm a PWM frekvencia

$$PR2 = \frac{2^N}{4} - 1 = \frac{1024}{4} - 1 = 255 \quad (20)$$

Láthatjuk, hogy így is ugyanazokra az eredményekre jutunk.

Most, hogy már rendelkezésünkre állnak a konfigurációhoz szükséges értékek, nézzük meg,

hogyan állítjuk be a PWM modult szoftveresen.

Az 53. ábrán láthatjuk PWM-el kapcsolatos függvényeket. Az inicializálást a `pwm_init` függvény végzi. Az inicializáció lépései a következők:

- RC1/CCP2 láb irányának beállítása kimenetnek.
- Timer2 előosztó beállítása
- TMR2 számláló regiszter törlése
- PWM frekvencia beállítása a PR2 regiszterrel
- PWM kitöltési tényező beállítása
- CCP2 modul indítása PWM üzemmódban
- Timer2 számlálásának indítása

```
25  int pwm_init(void)
26  {
27      TRISCbits.TRISC1 = 0;           //C port első bit kimenet a PWM miatt
28      T2CON = 0x00;                  //Timer2 előosztó 1:1
29      TMR2 = 0x00;                   //Timer2 számláló regiszter törlése
30      PR2 = 0xFF;                    //Timer2 periódus regiszter értéke 255
31      CCPR2L = 0x40;                 //PWM kitöltési tényező 25% (~700 RPM)
32      CCP2CONbits.CCP2X = 0;
33      CCP2CONbits.CCP2Y = 0;
34      CCP2CON = 0x0C;                //CCP2 PWM módban üzemel
35      T2CONbits.TMR2ON = 1;          //Timer2 számlálás indítása
36  }
37
38  int pwm_duty(int duty)
39  {
40      CCPR2L = duty>>2;
41      CCP1CON &= 0xCF;
42      CCP1CON |= (0x30&(duty<<4));
43  }
```

53. ábra. PWM beállításához használt függvények

A `pwm_duty` függvénnyel lehet beállítani a kitöltési tényezőt. A függvény hívásakor át kell adni neki a szabályozó által kiszámított beavatkozó jellel módosított kitöltési tényező értéket. A függvény ezt az értéket maszkolással elhelyezi a CCPR2L regiszterbe és a CCP1CON 4. és 5. bitjében.

A 4 bites kommunikáció miatt az adatokat és a parancsokat két részben kell elküldeni az LCD-nek. Az `Lcd_Cmd` és az `Lcd_Chr_CP` függvények ezt maszkolással oldják meg. A két függvény között csupán annyi a különbség, hogy az elsővel parancsot küldhetünk, a másodikkal pedig karaktert. Az 55. ábrán az `Lcd_Cmd` függvény látható.

4.8. Az USART modul programozása

A PIC16F877 vezérlő rendelkezik egy USART (Universal Synchronous Asynchronous Receiver Transmitter) modullal, melynek segítségével soros port-i kommunikációt tudunk létesíteni két mikrovezérlő között, vagy egy mikrovezérlő és egy PC között. Ennek hardveres részéről már a korábbi fejezetekben részletesen írtam. Ebben a fejezetben a kommunikáció mikrovezérlő felőli oldaláról szeretnék írni, valamint ennek szoftveres implementációjáról. USART modul használatával egyszerre küldhetünk és fogadhatunk is adatokat. Mivel én ebben a projektben csak adatküldésre szeretném használni az USART modult, ezért csak a küldő oldalt mutatom be.

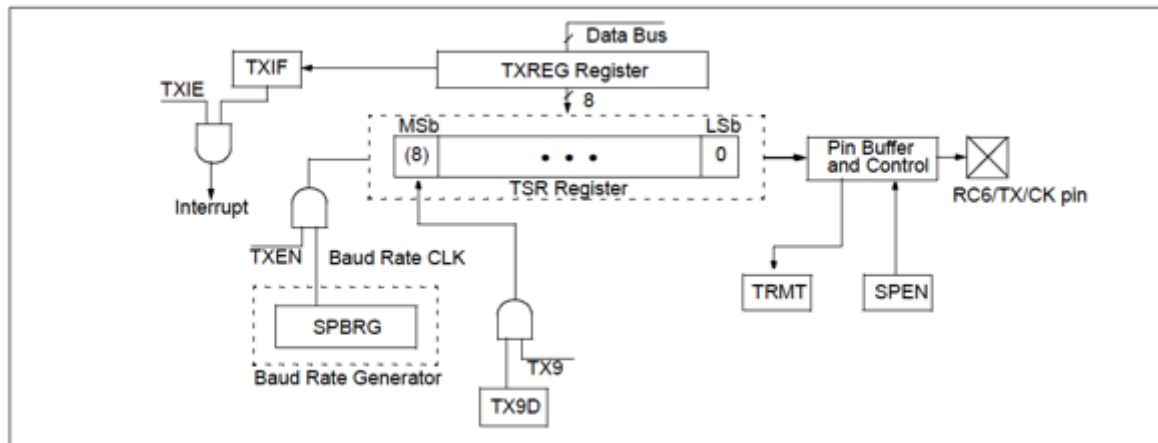
Ha az USART modult küldésre szeretnénk használni, szükségünk lesz a TXSTA konfigurációs regiszter helyes beállítására. Ennek a regiszternek a belső felépítése az x. táblázatban látható.

16. táblázat. A TXSTA regiszter felépítése

Bit 0	TX9D	Paritásbit engedélyezés	0	Nincs paritásbit
			1	Használunk paritásbitet
Bit 1	TRMT	Átvitel állapot jelzőbit	1	Átvitel folyamatban
			0	Átvitel vége
Bit 2	BRGH	Magas baud rate választás	0	Alacsony sebesség
			1	Magas sebesség
Bit 3	-			
Bit 4	SYNC	USART mód választó bit	0	Szinkron mód
			1	Aszinkron mód
Bit 5	TXEN	Átvitel engedélyező bit	0	Nincs átvitel
			1	Átvitel engedélyezve
Bit 6	TX9	9 bites átvitel engedélyezés	0	8 bites átvitel
			1	9 bites átvitel
Bit 7	CSRC	Órajel választó bit	0	Slave mód (szinkron)
			1	Master mód (szinkron)

A TXSTA regiszter beállításán kívül engedélyezni kell a soros port használatát az SPEN bit 1-re

állításával, valamint fel kell tölteni az SPBRG regisztert a használni kívánt baud rate-hez tartozó értékkel. A PIC16F877 katalógusa tartalmaz egy táblázatot, aminek segítségével meghatározhatjuk, hogy az általunk használt órajel frekvenciához, és a használni kívánt baud rate-hez milyen SPBRG érték tartozik.



57. ábra. Az USART küldő modul blokkvázlata (forrás: [7])

Az 57. ábrán látható az USART küldő oldal blokkvázlata. Az adatküldés menete a következő:

- Felkonfiguráljuk az USART modult a TXSTA és az SPBRG beállításával, valamint az SPEN bit 1-re állításával.
- Engedélyezzük a küldést a TXEN bit 1-re állításával.
- Betöltjük az adatot a TXREG regiszterbe.
- A TXREG tartalma átkerül a TSR regiszterbe, a TXIF értéke 1 lesz.
- A kiválasztott baud rate órajellel megkezdődik a TSR tartalmának küldése.
- Ha a TSR regiszter minden bitje el lett küldve a TRMT értéke 1 lesz.
- Ha van betöltve új adat a TXREG-be, akkor az átkerül a TSR-be, majd indul a küldés feljebb leírtak szerint.

Az LCD-hez hasonlóan az USART modulhoz is rengeteg előre elkészített libraryt találhatunk az interneten, sőt a Microchip application library-ban is található USART kezelő függvény csomag. Én ebben az esetben is egy leegyszerűsített, csak küldésre alkalmas library-t használok.

```

6 void USARTInit(unsigned int baud_rate)
7 {
8
9     switch(baud_rate)
10    {
11        case 9600:
12            SPBRG=129;
13            break;
14        case 19200:
15            SPBRG=64;
16            break;
17        case 28800:
18            SPBRG=42;
19            break;
20        case 33600:
21            SPBRG=36;
22            break;
23    }
24
25    TXSTAbits.TX9=0; //8 bit transmission
26    TXSTAbits.TXEN=1; //Transmit enable
27    TXSTAbits.SYNC=0; //Async mode
28    TXSTAbits.BRGH=1; //High speed baud rate
29    RCSTAbits.SPEN=1; //Serial port enabled
30
31 }

```

58. ábra. USART modul inicializáló függvény

A függvény hívásakor át kell adnunk a használni kívánt baud értéket. A switch/case szerkezet segítségével a program kiválasztja a megfelelő SPBRG értéket. Ezután beállítjuk a 8 bites átvitelt, aszinkron módot, nagy sebességű baud ratet, és engedélyezzük a soros port-i átvitelt.

Mivel itt is karaktereket tudunk küldeni, a küldésre használt függvények nagyon hasonlóak az LCD-hez használtakhoz.

```

33 void USARTWriteChar(char ch)
34 {
35     while(!PIR1bits.TXIF);
36     {
37
38     }
39     TXREG=ch;
40 }
41
42
43 void USARTWriteString(const char *str)
44 {
45     while(*str!='\0')
46     {
47         USARTWriteChar(*str);
48         str++;
49     }
50 }

```

59. ábra. USART küldő függvények

5. Összefoglalás

A dolgozat zárásaként szeretnék a jövőbeli tervekről és lehetőségekről írni pár mondatot. Ez a dolgozat az eszköz megtervezéséről szólt, mivel a leadási határidőig nem készült el a hardver építése. Szakdolgozatom témája ugyan ez a demonstrációs eszköz, annyi kiegészítéssel, hogy ott már az eszköz építése és tesztelése is megvalósításra kerül.

A közeljövőben tervbe van véve, hogy a megépített eszközhez a PIC mikrokontroller helyett az NI Labview fejlesztőkörnyezetével tervezzünk és valósítsunk meg szabályozót. Ezzel az eszköz modularitását, újrahasznosíthatóságát lehet demonstrálni.

Ennek a dolgozatnak a keretei között a HMI modult csak a mért és beállított értékek megjelenítésére használok, azonban rengeteg fejlesztési lehetőség van még a modulban. A jövőben szeretnék a HMI modul felhasználásával egy olyan strukturált kezelő felületet létrehozni amelyen keresztül a programkód átírása nélkül módosíthatók a szabályozó paraméterek, választhatók ki a használni kívánt szabályozó algoritmusok.

A tervezési szakasz elején felmerült egy örvényáramú fék építése fejlesztése és megépítése, amely a motor terheléseként használható. A jövőben ennek megvalósítása is érdekes feladat lehet.

Jelenlegi állapotában az eszköz csak fordulatszám szabályozásra alkalmas, viszont szoftveres módosításokkal lehetőség lenne pozíció szabályozás megvalósítására is.

Úgy gondolom, hogy az általam tervezett demonstrációs eszköz több szempontból is hasznos kelléke lesz a tanszéki laboratóriumnak, mivel egyrészt felhasználható oktatási célokra, a szabályozók bemutatására, a programozási és szabályozástechnikai ismeretek elmélyítésére, másrészt pedig az érdeklődő diákok egy jó alapról elindulva tovább fejleszthetik az eszközt TDK, illetve szakdolgozat/diplomamunka keretei között. A tanszék az ilyen eszközöknek köszönhetően sokkal gyakorlat orientáltabbá teheti a szabályozástechnika oktatást, fejlesztheti laborok felszereltségét, olyan módon, hogy ebben mi diákok is részt tudunk vállalni.

Irodalomjegyzék

- [1] Tüü-Szabó Boldizsár: *Egyenáramú motor modellezése Matlab-ban és fordulatszám szabályozása mikrokontrollerrel*
TMDK-dolgozat, Győr (2013).
- [2] Transtecno: *Permanent Magnets D.C. Electric Motors*
Gyártói katalógus, és adatlapok
- [3] International Rectifier: *IRFZ34N HEXFET, Power MOSFET datasheet*
Gyártói alkatrész adatlap
- [4] Motorola: *MC78T00 Three-Ampere Positive Voltage Regulators, Semiconductor technical data*
Gyártói alkatrész adatlap
- [5] Maxim Integrated Products: *Low-Cost, Precision, High-Side Current-Sense Amplifier*
Gyártói alkatrész adatlap, 19-1184; Rev0; 12/96
- [6] Microchip: *DC Motor Control Tips 'n Tricks*
Segédlet, 2008, 5 oldal
- [7] Microchip: *PIC16F87X 28/40-Pin 8-Bit CMOS FLASH Microcontrollers, DS30292D*
Gyártói alkatrész adatlap
- [8] Sharp: *GP1A05/GP1A22LC/GP1A23LC/GP1A25LC, OPIC Photointerrupter with Connector*
Gyártói alkatrész adatlap
- [9] Maxim Integrated Products: *+5V-Powered, Multichannel RS-232 Drivers/Receivers*
Gyártói alkatrész adatlap, 19-4323; Rev9; 4/00
- [10] Bolymin, Inc: *LCD Modules Specification, BC1602E series*
Gyártói alkatrész adatlap
- [11] Keviczky László (, Bars Ruth, Hetthéssy Jenő, Barta András, Bányász Csilla):
Szabályozástechnika
UNIVERSITAS-GYŐR Kht, Győr (2006), 410 oldal
- [12] Szücs Zoltán: *A digitális PID szabályozó (oktatási segédanyag)*
Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest (2008), 15 oldal
- [13] Bali Csaba: *Robot-szabályozási algoritmusok kísérleti vizsgálta*
Dolgozat, Budapesti Műszaki és Gazdaságtudományi egyetem, Budapest (2011), 61 oldal