Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Rendszerterv készítése a következő fejlesztési projekthez: „Rapid prototyping lehetőségeinek vizsgálata beágyazott autóipari környezetben”

Rendszertervezés (VIMIM238) házi feladat
2011/2012. I. félév

Sipos-Takáts Bence László (T35NOC)

I. évf., villamosmérnök szakos hallgató
MSc Beágyazott információs rendszerek szakirány

2011. december 1.

Tartalom

1. Bevezetés

Önálló laboratóriumi feladatomat a Robert Bosch Kft-nél végeztem Budapesten, mivel év eleje óta itt dolgozom. Az osztály, ahol vagyok, váltóvezérlőkkel foglalkozik, főképp fejlesztéssel, teszteléssel és szériatámogatással. Mint a legtöbb autóipari nagyvállalatnak, a Bosch fejlesztési folyamata nehézkes és időigényes. Ennek oka a vevők által megkövetelt minőség biztosítása és a validált fejlesztési folyamathoz való ragaszkodás. Emiatt lassan reagál az ipar változásaira, előrehaladására. Természetesen ez nem azt jelenti, hogy le van maradva a korral, hanem hogy az újítások csak évekkel később jelennek meg. Itt nem csak a termékbeli újításokra kell gondolni, hanem technológiai előrelépésekre is.

Jelenleg egy termék fejlesztéséhez, vagyis ahhoz, hogy az ötlet kitalálásától a gyártósoron sorozatban futó termék legyen, körülbelül két-három év szükséges. Azonban egyre jobban érezhető, hogy ez az idő túl hosszú, nem tartható. Útelágazáshoz érkezett a fejlesztés, vagy marad a réginél, vagy változtat a fejlesztési folyamaton. Mindkettőnek megvan a maga előnye. A régi fejlesztés azt az elvet vallja, hogy „ami bevált, azon minek változtatni”. Ez azonban érezhető, hogy nem tartható, hiszen aki nem halad a korral, előbb-utóbb elbukik. Főleg az autóiparban, hogy a verseny kiélezett és nagyon magas minőségi kritériumoknak kell megfelelni. Nagy előnye azonban, hogy a vevők ezt már elfogadták, validálták, és ők is kicsit rettegnek az új dolgoktól.

Másik lehetőség egy új fejlesztési folyamat bevezetése, név szerint a Model-based Desing, azaz modell alapú fejlesztés, azon belül is a Rapid Prototyping. Önálló laboratóriumi munkám ennek megismeréséről és lehetőségeinek felkutatásáról szól, valamint arról, hogy hogyan lehetne ezt integrálni a Bosch fejlesztési folyamataiba úgy, hogy a fejlesztés közbeni dokumentációk, output-ok és mérföldkövek megmaradjanak. Így a vevők kapják meg, mint a hagyományos fejlesztés alatt.

Az önálló laboratóriumom és diplomamunkám során kifejlesztek és véghezviszek saját fejlesztési procedúrát egy 8 bites mikrokontrollerrel meghajtott szénkefe nélküli DC (BLDC) motorhoz. Gyakorlatilag ezen mutatom be, illusztrálom a való életben a fejlesztési folyamatot, minden egyes lépésen ténylegesen végigmenve. Házi feladatomban ennek projekt- és rendszertervét, követelmény- és projekt konfiguráció menedzsmentjét, valamint dokumentáció generálását fogom elkészíteni.

2. Projekt terv

A projekt terv elkészítéséhez az ingyenesen használható Openproj nevezetű programot használtam, melyben egyszerűen lehet munkaszakaszokat definiálni, hozzájuk felelőst rendelni, valamint időköltségeket megadni és ez alapján a kritikus utat szinte automatikusan legenerálni. A munkaszakaszok hosszát és egymásra épülését időben ábrázolja (Gantt diagram), így könnyedén át lehet tekinteni a projektet. Továbbá a megfelelő órabérek megadása után nemcsak a fejlesztésben részt vevő erőforrások büdzséjét, hanem az egyes szakaszok költségét is ki lehet számolni, ezt összegezve pedig a projekt teljes költségvetését.

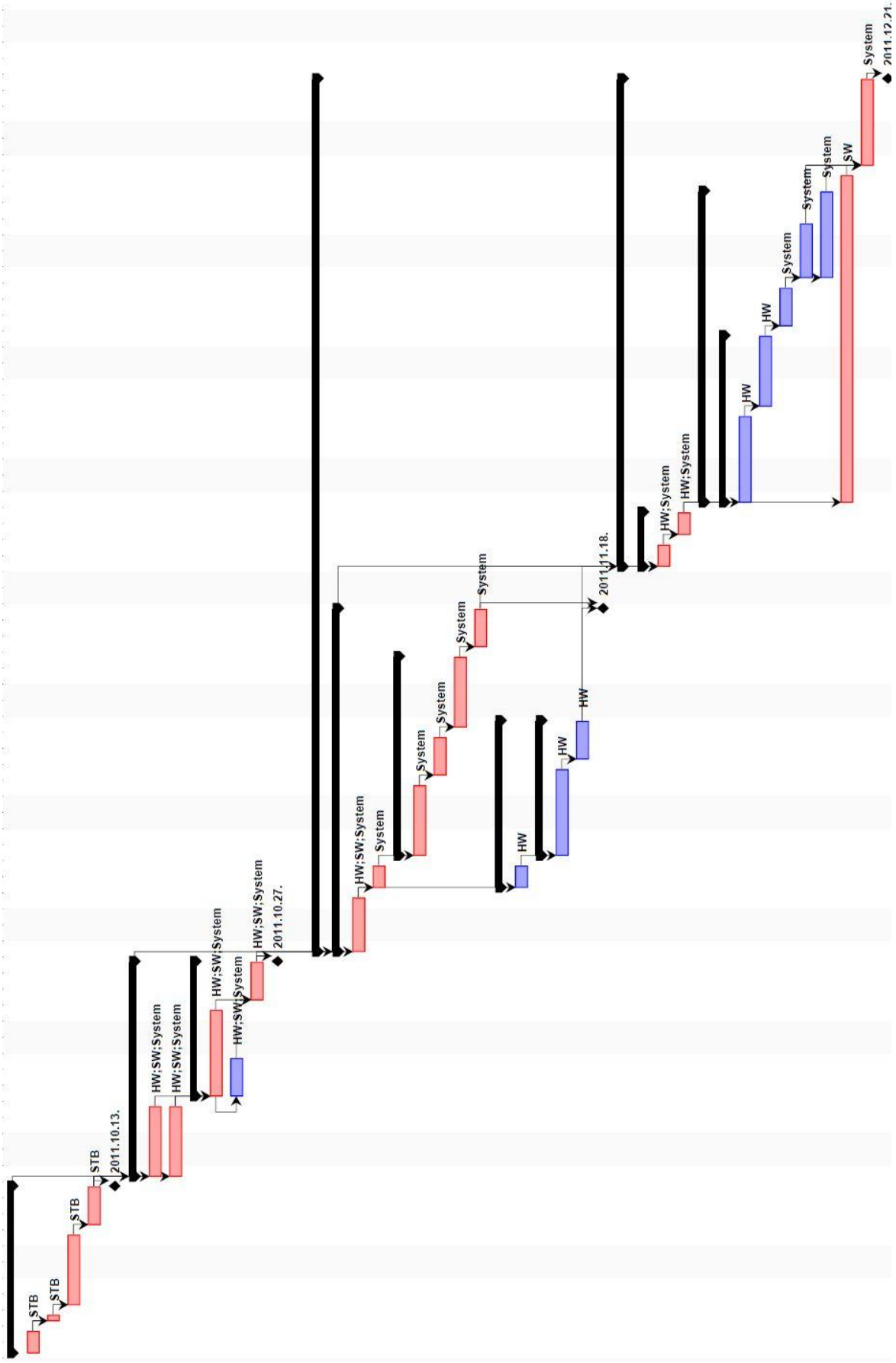
A program használatakor először meg kell határozni a munkaszakaszokat, majd az „elődeit". Ez itt azt jelenti, hogy azon szakaszokat kell megadni, amelyek megelőzik azt a szakaszt. Így lehet időben egymásra épülővé tenni a fejlesztést. Az elődi kapcsolatok többfélék lehetnek, kezdve az „együtt kezdés"-sel, az „együtt befejezés"-en át, a „nem kezdődhet, vagy nem fejeződhet be előbb, mint a másik folyamat"-ig. Ezt követően kitűzzük a projekt kezdésének idejét, és meghatározzuk az időköltségeket. Ez általában korábbi tapasztalatok, vagy hasonló folyamatok költségein alapul. Mivel én nem foglalkoztam még ilyen jellegű fejlesztéssel, racionálisan végiggondoltam, mennyi idő szükséges az egyes munkákra, és ez alapján definiáltam az időráfordítások hosszát.

A következő lépés a folyamatok erőforrásokhoz rendelése. Először azonban definiálni kell az erőforrásokat, azaz a munkát elvégző embereket. Meg lehet adni feladatkört, munkatípust, órabéreket, napi óraszámot, munkarendet és még sok más információt. Eztán hozzá kell rendelni az egyes feladatokhoz az egyes embereket, azaz kinevezni a munkaszakaszok felelőseit. Én négy embert definiáltam, ami vázlatosan mutatja, milyen feladatkörökre van szükség. Első egy System Designer, aki a rendszer tervezését és az egész rendszert látja át. Második egy Hardware Designer, aki a hardveres feladatok megtervezését és végrehajtásáért felelős. Továbbá szükség van egy Software-esre is, aki a kód generálásáért és a platform megteremtéséért, felügyeletéért felelős. Alapesetben ez a három munkacsoport elegendő a fejlesztéshez. Viszont az önálló laboratórium során a fejlesztést megelőzi irodalomkutatás, valamint maga a process kitalálása, így erre definiáltam magam (STB), mint negyedik ember. Ezáltal nyomon követhető a teljes önálló laboratórium, majd diplomamunka során a szakaszt végző erőforrás kiléte. Természetesen az első három ember is én leszek, de szükséges lebontani külön csoportokra, hiszen egy valódi fejlesztés szimulációja a cél.

A következő ábrákon bemutatom magát a munkafolyamatokat és a Gantt diagramot. Az optimális láthatóság érdekében a diagram elforgatva található, valamint más időosztást alkalmaz. A táblázat a heti 20 órás gyakornoki időbeosztásom alapján kalkulált, belevéve az iskola-, és vizsgaidőszakot, azonban kisebb mérete miatt a Gantt heti 40 órás, szabadság nélküli, teljes munkaidős esetre lett számolva.

WBS	Name	Duration	Start	Finish	Predeces...	Resource Names
1	**TASK#1**	**22 days**	**2011.10.03. 8:00**	**2011.11.01. 17:00**		
1.1	RB PEP megismerése	2 days	2011.10.03. 8:00	2011.10.06. 17:00		STB
1.2	Mit használnak mások?	1 day	2011.10.07. 8:00	2011.10.11. 17:00	2	STB
1.3	RP lehetőségek megismerése	3 days	2011.10.12. 8:00	2011.10.20. 17:00	3	STB
1.4	Saját fejlesztési process definiálása	3 days	2011.10.21. 8:00	2011.11.01. 17:00	4	STB
	Milestone#1	0 days	2011.11.01. 17:00	2011.11.01. 17:00	5	
2.1	**TASK#2**	**25 days**	**2011.11.02. 8:00**	**2011.12.06. 17:00**	**1**	
2.2	BLDC motorok megismerése	7 days	2011.11.02. 8:00	2011.11.10. 17:00	5	HW;SW;System
2.3	BLDC motorok lehetséges vezérlési módsz. megism.	7 days	2011.11.02. 8:00	2011.11.10. 17:00	5	HW;SW;System
2.4	**Rapid Prototiping Tool-ok megismerése**	**18 days**	**2011.11.11. 8:00**	**2011.12.06. 17:00**		
2.4.1	MATLAB	10 days	2011.11.11. 8:00	2011.11.24. 17:00	8;9	HW;SW;System
2.4.2	Simulink	8 days	2011.11.11. 8:00	2011.11.22. 17:00	11SS	HW;SW;System
2.4.3	Saber	8 days	2011.11.25. 8:00	2011.12.06. 17:00	11;12	HW;SW;System
	Milestone#2	0 days	2011.12.06. 17:00	2011.12.06. 17:00	13	
	Fejlesztés	**96 days**	**2012.02.14. 8:00**	**2012.06.26. 17:00**		
3	**TASK#3**	**38 days**	**2012.02.14. 8:00**	**2012.04.05. 17:00**	**7**	
3.1	Requirement management	3 days	2012.02.14. 8:00	2012.02.16. 17:00	13	HW;SW;System
3.2	Tesztek megtervezése	2 days	2012.02.17. 8:00	2012.02.23. 17:00	17	System
3.3	Motor modell ill. paraméteridentifikáció	3 days	2012.02.24. 8:00	2012.03.06. 17:00	18	System
3.4	Kapcsoló logika megvalósítása Simulinkben	3 days	2012.03.07. 8:00	2012.03.15. 17:00	19	System
3.5	Vezérlő algoritmus terv. és implementálása	3 days	2012.03.16. 8:00	2012.03.27. 17:00	20	System
3.6	Tesztelés	3 days	2012.03.28. 8:00	2012.04.05. 17:00	21	System
3.7	**SwIL-hez HW tervezése és megépítése**	**23 days**	**2012.02.17. 8:00**	**2012.03.20. 17:00**		
3.7.1	Specifikáció	2 days	2012.02.17. 8:00	2012.02.23. 17:00	17	HW
3.7.2	**HW tervezés**	**18 days**	**2012.02.24. 8:00**	**2012.03.20. 17:00**		
3.7.2.1	Kapcsolási rajz	4 days	2012.02.24. 8:00	2012.03.08. 17:00	24	HW
3.7.2.2	NYÁK terv	3 days	2012.03.09. 8:00	2012.03.20. 17:00	26	HW
	Milestone#3	0 days	2012.04.05. 17:00	2012.04.05. 17:00	22;27	
4	**TASK#4**	**58 days**	**2012.04.06. 8:00**	**2012.06.26. 17:00**	**16**	
4.1	**SwIL végrehajtása**	**10 days**	**2012.04.06. 8:00**	**2012.04.19. 17:00**		
4.1.1	Eredmény kiértékelése	5 days	2012.04.06. 8:00	2012.04.12. 17:00	22;27	HW;System
4.1.2	Végleges HW specifikációja	5 days	2012.04.13. 8:00	2012.04.19. 17:00	31	HW;System
4.2	**Végleges HW tervezése és megépítése**	**35 days**	**2012.04.20. 8:00**	**2012.06.07. 17:00**		
4.2.1	**HW tervezés**	**18 days**	**2012.04.20. 8:00**	**2012.05.15. 17:00**		
4.2.1.1	Kapcsolási rajz	4 days	2012.04.20. 8:00	2012.05.03. 17:00	32	HW
4.2.1.2	NYÁK terv	3 days	2012.05.04. 8:00	2012.05.15. 17:00	35	HW
4.2.2	Leszimulálás Saber-ben	3 days	2012.05.16. 8:00	2012.05.24. 17:00	36	System
4.2.3	WC analízis	2 days	2012.05.25. 8:00	2012.05.31. 17:00	37	System
4.2.4	Algoritmus finomítás	4 days	2012.05.25. 8:00	2012.06.07. 17:00	37	System
4.3	Procira automatikus kódgenerálás biztosítása	15 days	2012.04.20. 8:00	2012.06.12. 17:00	32	SW
4.4	Kész HW + SW tesztelése	4 days	2012.06.13. 8:00	2012.06.26. 17:00	38;39;40	System
	Milestone#4	0 days	2012.06.26. 17:00	2012.06.26. 17:00	41	

1.1. ábra: Munkafolyamatok, időköltségeik és felelőseik



1.2. ábra: Gantt diagram

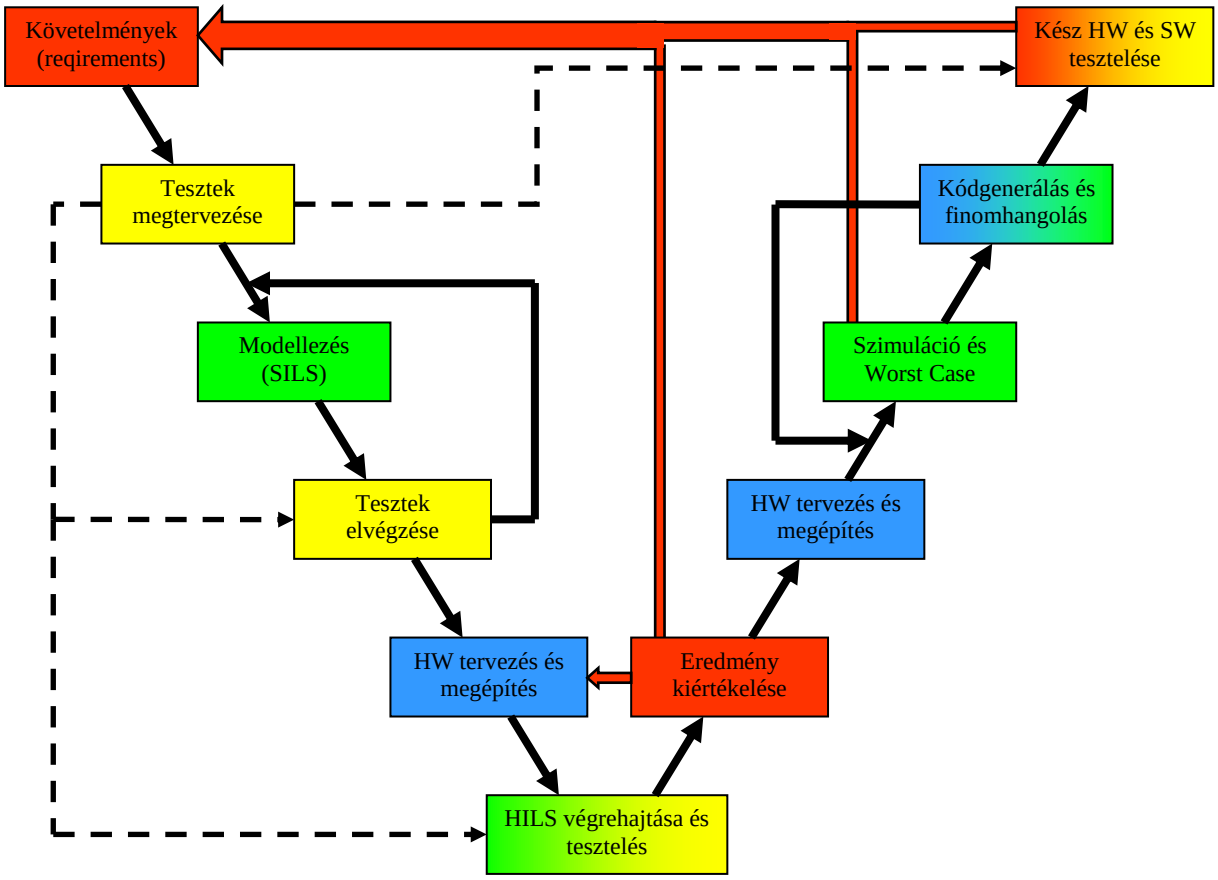
Látható, hogy a folyamat 4 fő részből áll. Az első az említett irodalomkutatás, valamint a fejlesztés kitalálása. A második a fejlesztő eszközök (tool-ok), valamint a megoldandó probléma megismerése. Jelen esetben a BLDC motorokban kellett elmélyednem, valamint a Matlab Simulink és Saber programok használatában. A Task 2-re nincs szükség, ha már korábban is volt hasonló fejlesztés, vagyis a háttértudás, tool-ok ismerete adott. Ekkor a projektbe csak a hármas és négyes Task tartozik. Ezt jelenleg nem részletezném, hiszen a fejlesztési folyamatról részletesen beszélek a rendszerterv során.

A projektterv összegzéseképp elmondanám, hogy a pirossal jelölt kritikus út alapján 2012 júniusára elkészülök, azonban ezt befolyásolja, hogy munkám során nem tudok mindig a diplomamunkámmal foglalkozni, valamint az időfüggvények erős becslések. Azonban ha tartom magam hozzá, jó eséllyel be tudom fejezni időre a diplomamunkám.

2. Rendszertervek

2.1 Fejlesztés menete, iterációk

A rendszertervben először – az előzőleg említett – Taskok egymásra épülését, iterációs lehetőségeit mutatnám be, majd a hardveres (technikai) és a szoftveres rendszerterveket.



2.1. ábra: Fejlesztés menete, iterációk

A különböző színekkel próbáltam érzékeltetni a különböző típusú munkákat. Eszerint a piros a dokumentációval kapcsolatos, sárga a tesztekkel, teszteléssel, zöld a modellezéssel, valamint kék pedig a tervezéssel és építéssel.

A folyamat lényege, hogy a követelmények meghatározása után megtervezzük azon teszteket, mellyel később mind a modellünket, mind az elkészült prototípusunkat tesztelni tudjuk – ezen kapcsolatot a szaggatott vonal jelképezi. Ezt követően lemodellezzük az elkészítendő vezérlést (szabályozókört), valamint a szakaszt (plant-et): jelen esetben megszerkesztjük a motor és a motorvezérlő modelljét. Ezután elvégezzük rajta a teszteket (Software-in-the-Loop Simulation, SILS) és addig finomítjuk a modellünket, amíg nem

kapjuk a legjobb eredményt. Itt nem csak a legjobb szabályzókör, valamint a paraméterek beállítására kell gondolni, hanem magára a modellre is. Ez az első iterációs lépés, melyet a visszafelé mutató fekete nyíllal jelöltem.

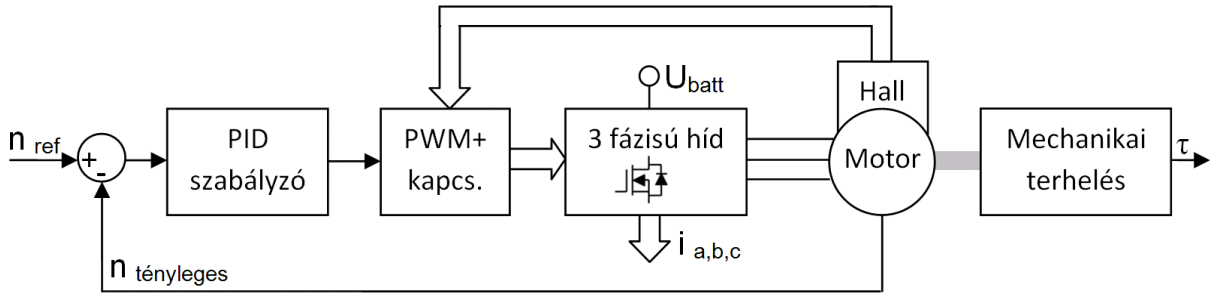
Amint kész a lehető legvalósághűebb modellünk valós paraméterekkel, következik az első prototípus megépítése. Ez azonban nem a végleges prototípus, általában egyszerű és könnyen realizálható, így fajlagos költsége alacsony. Célja a Hardware-in-the-Loop Simulation (HILS) elvégzése, melynél már rendelkezünk hardverrel, de vagy a szakasz, vagy a vezérlés modellje még számítógépen található: a hardver ezzel van real-time kapcsolataban, kommunikál vele, vezérli azt vagy vezérlést kap tőle. Az én esetemben ennél a tesztnél a valós motort szeretném tesztelni a vezérlés modelljével, tehát a hardvernek tartalmaznia kell a motormeghajtó blokkot, valamint a jelek és megfelelő vezetékek kivezetéseit (I/O portok). A tesztelés végrehajtása után az eredményeket ki kell értékelni, és össze kell vetni a mind az első prototípus specifikációjával, mind a követelményekkel (ezt mutatják a piros nyilak). A következmények levonása után, az előző hardware design-t felhasználva meg kell tervezni és építeni a végleges kapcsolási rajzot és NYÁK tervet. Ha ezzel készen vagyunk, az egészet le kell szimulálni kapcsolásszimuláló programban (Saber), ahol már a valódi alkatrészeket, valós karakterisztikákat alkalmazunk. Ezáltal el tudjuk végezni a worst case szimulációkat egyrészt automatikusan, másrészt olyan esetekre is, amit nehéz, vagy veszélyes lenne megvalósítani.

Ez idő alatt a szimulációkban létrehozott vezérlésmodellt le kell fordítani a target processzorra. A szimulációs programok nagyrészt támogatja ezt. Azonban az ilyen fordításoknál a legtöbb esetben a valós hardver nem úgy viselkedik, mint a szimulált modell, ezért a gyártásképes eszköz elkészülése előtt még több finomhangolással egybekötött iterációs lépés szükséges. Ezt ismét a fekete visszafelé mutató nyíl illusztrálja. Természetesen, ha megváltoztatunk valamit, az analízist is újra kell futtatnunk, hogy a worst case esetek a módosított paraméterekre is reálisak legyenek, és hogy a követelményeknek megfeleljenek. Végezetül az elkészült hardvert és a beleimplementált szoftvert együttesen teszteljük és verifikáljuk.

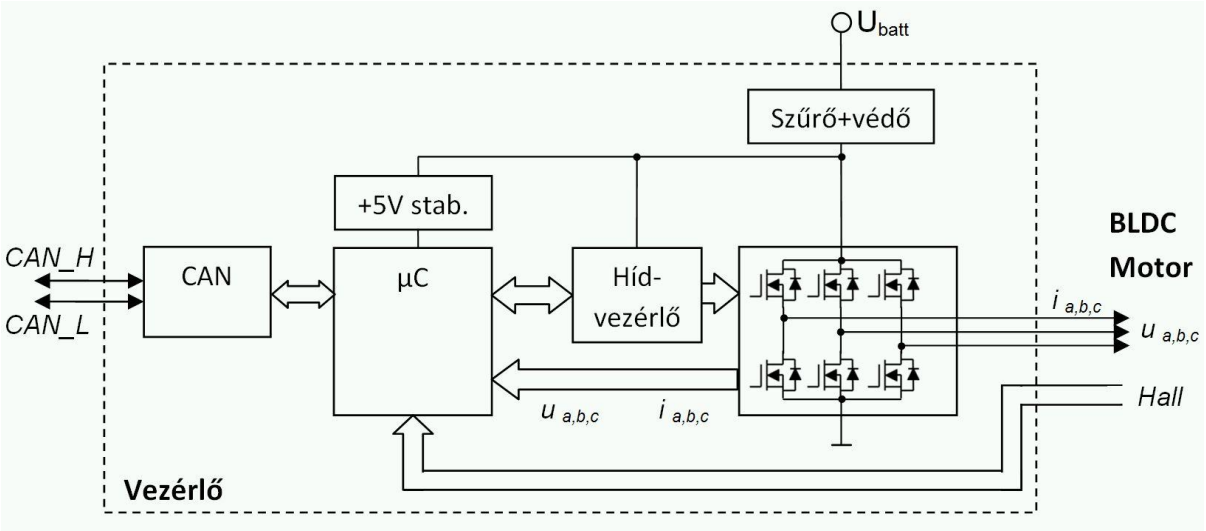
Ha megfelel a követelményeknek, specifikációnak, tesztelve, validálva és verifikálva lett, a fejlesztési folyamatunk ezen része véget is ért, elkezdődhet a gyártás. Természetesen itt nem áll meg a tesztelés: a hosszú, tartóssági tesztek még vissza vannak.

2.2 Logikai és Technikai Rendszerterv

A 2.2. ábrán látható SILS modell terve, az 2.3-on az ebből megvalósítható hardver blokkvázlata, ami a logikai és technikai rendszerterv.



2.2. ábra: Modell rendszerterve



2.3. ábra: Hardver rendszerterve

Látható, hogy egy impulzusszélesség modulációval (PWM) szabályozott háromfázisú híd vezérli a motort, melynek kimenő fordulatszámából egy PID szabályzó állítja elő az impulzus szélességét. Továbbá a motor Hall-szenzorjai alapján irányított kapcsoló modul állítja elő a megfelelő kommutációt és annak idejét.

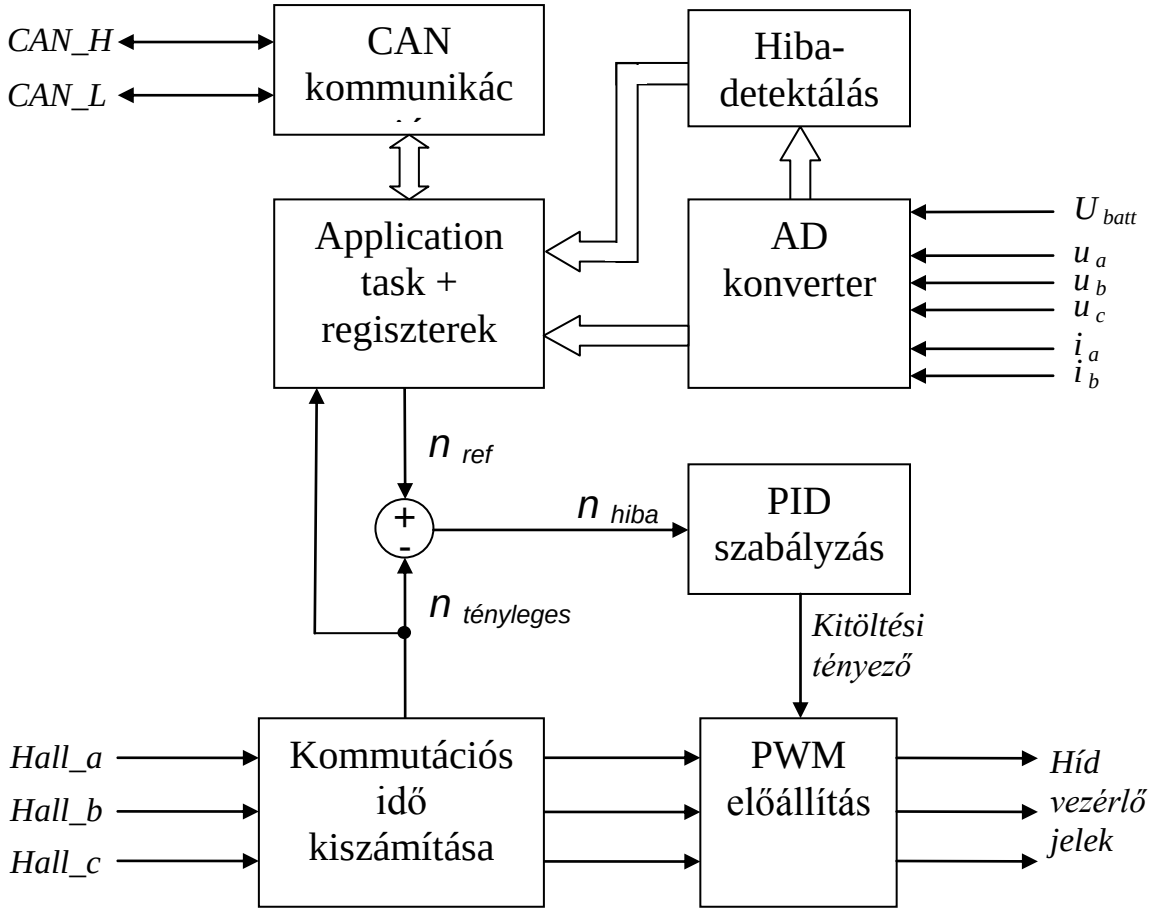
Ez hardveresen úgy valósul meg, hogy egy mikrokontroller elvégzi a PID szabályozást, a PWM előállítását és a kommutálási idő kiszámítását a motor visszaolvasott Hall jelei alapján. CAN-en csatlakozik a környezetéhez, ezzel létrehozható egy stand-alone eszköz, azaz CAN-en csak üzeneteket kap, és arra reagál vagy a motor vezérlésének megváltozásával, vagy akár egy számított paraméter, mint például az AD konverterrel

visszaolvasott fázisáram vagy feszültség visszaküldésével. Szükség van még stabil, szűrt tápfeszültségre és védelemre mind a mikrokontrollernek, mind az egész eszköznek.

A hídvezérlő a mikrokontroller PWM jelei alapján elvégzi a tényleges kommutációt, valamint minimális hibajelzéssel rendelkezik. Ez a blokk hajtja meg a FET-eket, melyek pedig meghajtják a motort.

2.3 Szoftver architektúra és modul terv

A szoftver architektúra a 2.4. ábrán látható, azonban ez vázlatos, hiszen a fejlesztési koncepció alapján a szoftverkód automatikusan generálódik a modellből. Gyakorlatilag ez található a mikrokontrollerben.



2.4. ábra: Szoftver architektúra

Az egésznek a lelke az Application task, ami úgy képzelhető el, mint egy felső réteg a programban. Ez végzi az alsó szintű vezérléseket, mint például a referencia fordulatszám

beállítása, vagy a beolvasásokat, vagyis az AD konverter értékeit. Lekezeli a CAN kommunikáció felső rétegét, azaz a CAN modul által fogadott parancsokat értelmezi, vagy a megfelelő adatokat továbbadja, elküldi a modulnak. Ebbe beleírtam jelképesen a regisztereket, hiszen minden változó (paraméterek, visszaolvasott adatok) itt tárolódnak. Továbbá rendelkezik hibadetektálással (túláram, zárlat, vezetékleszakadás), és ez alapján úgymond megvédi magát.

Ahogy korább is mondtam, a beolvasott Hall jelek alapján kiszámolja a kommutációs időt, mely gyakorlatilag a hídban található FET vezérlők kapcsolási ideje. Ezek a jelek lesznek PWM-mel szabályozva, melyet a hiba értéke alapján a PID szabályozó állít elő.

3. Követelmény menedzsment

Mivel ez egy autóipari alkalmazás, az autóipari előírásoknak meg kell felelnie. Alapvetően a vevővel szükséges egyeztetni a követelmények specifikálásáról, azonban önálló laboratóriumi fejlesztésem során e hiányában minél általánosabb eszközt próbáltam létrehozni. Az egyszerűbb áttekintés érdekében táblázatos formában sorolom fel.

Tápfeszültség		
	működési tartomány (Vbatt)	*+ 9 és 16V*
	Fordított polaritás	*-14V*
	Túlfeszültség	*>18V*
	ESD védelem	*±4kV*

Áram		
	Maximális (motornak)	*10A rms*
	Nyugalmi áram (kikapcsolt állapot)	*1mA*
	Soft-start	

Vezérlési módok		
	RPM vezérlés	*100-5000RMP, 1%*
	Nyomaték (áram) vezérlés	*1-10A rms, 1%*
	Kitöltési tényező (open-loop)	*0-100%*

Kommunikáció		
	CAN interface (fogadás)	*Start üzenet*
		Stop üzenet
		Vezérlési mód beállítás
		Paraméter beállítás
	CAN interface (küldés)	*Paraméterek, értékek*

Diagnosztika		
	Motor terhelés monitorozása	*túláram, mech. terhelés*
	Hibás bejövő adat lekezelése	*CAN üzenet*
	Forgásirány, rotor pozíciódetektálás	*motor beakadás észlelés*
	Fázis rövidzár, szakadás detektálása	
	Háromfázisú híd hibadetektálása	

Zavarvédelem		
	Tápfeszültség ingadozás	*zavarjelek, szinusz*
	Tápfeszültség beesés	*indítás*

Hőmérséklet		
	működési tartomány	-40 és +140°C között

A követelmények az előző pontban mutatott módon (2.1. ábra), folyamatosan figyelembe vannak véve, hiszen a teszteket a követelmények alapján hozzuk létre, valamint a modellek, prototípusok mind ezen tesztek alapján lesznek validálva. Ezáltal a követelménynek való megfelelés az egész fejlesztést végigköveti. A tesztek eredményeiről készített dokumentumok tartalmazzák ezt.

Fontos az eredmények rögzítése, hiszen ez alapján lehet a további fejlesztést nyomon követni. Egy megfelelően ledokumentált projektben nemcsak a felmerülő kérdésekre adott megoldások okait lehet feltárni, hanem alternatív megoldásokat is beleírhatunk. Ez akkor előnyös, ha fejlesztésünkben zsákutcába kerülünk, azaz egy megoldásról később kiderül, hogy kevésbé, vagy egyáltalán nem teljesíti a követelményeket, könnyebb egy új alternatívát keresni.

4. Verziókövetés és szoftverdokumentáció

4.1 Verziókövetés

Az önálló laboratóriumom mostani részében még nem rendelkezem szoftverkóddal, csak modellekkel, így a verziókövetés nehezen megoldható. Azonban tudom rá alkalmazni az eddig jól bevált, saját magam által kifejlesztett módszert. Ennek a lényege két módosítási folyamatra vetíthető le, egyik a verzión belüli, másik a verziók közötti. Most modellekre fogom bemutatni.

Először létrehozok egy modellt, mondjuk modell_v1.0 néven, és ezt fejlesztem. Emellett létrehozok a mappában egy txt fájt, melyben a megvalósítom a verziókövetést. Ameddig fejlesztem a modellt, nem módosítom a nevét, ellenben minden egyes mérföldkőnél a txt-ben beírom a tulajdonságait és hiányosságait. Ezt egészen addig így csinálom, míg a modell a lehető legalapvetőbb funkciókkal, tulajdonságokkal nem rendelkezik, azaz lefordul és nagyjából azt csinálja, amiért létrehoztam. Ez a kiindulási modell, ehhez bármikor vissza lehet térni.

Ezután minden egyes változtatásnál létrehozom a korábbi modell másolatát, átnevezem (például v1.1-re) és azon dolgozok tovább. A másolat létrehozása biztosítja a mindenkori visszatérés lehetőségét. Továbbá a txt-ben is egy új bekezdést kezdek, ahol leírom mit javítottam ki az előző modellhez képest és mivel bővültek funkciói. Nagyobb változtatás, mondjuk konstrukció, működésbeli változtatás esetén a „ .v” utáni számot inkrementálom. Jelen esetben eddig két modellem van, egyik PI, másik már PID szabályozóval. Az első az 1.0-ás változat, a második az 1.1-es. Egyedüli különbség csak a szabályzó, azonban a txt-ben az is számon van tartva, melyik miben jobb és miért. Ezzel a módszerrel könnyen követhető nemcsak a verziók, hanem kronologikusan a fejlesztés menete is.

Megemlíteném még, hogy a modellen belüli változtatás nem mindig jár a txt megváltoztatásával, hiszen egy napi fejlesztés is elmehet úgy, hogy lényegében nem fejlődik a modell; azonban ha egy problémát kiküszöbölök, vagy egy jobb beállítást, paramétert találok, azt feljegyzésre kerül, sőt új verziót is kaphat. Természetesen ez nem azt jelenti, hogy ha az „A” változót 1.2-ről 1.3-ra változtatom, egyből új verziószámot kap, de például ha egy hibásan működő algoritmust kijavítok, az belekerül a txt-be és új verzió lesz.

4.2 Szoftverdokumentáció

A forráskód automatikusan lesz legenerálva a modellből, és mivel még közel sem tökéletes a modell, nincs értelme forráskódot fordítani belőle. Nem is tudnám hol futtatni, hiszen a prototípus elkészítése is csak következő félévben várható. Azonban a szoftverdokumentációról általánosságban szólnék egy-két szót.

Talán az egyik legjobban használható rendszer a Doxygen. Ez egy nyílt forráskódú szoftver, mely leegyszerűsíti és automatizálja a projektdokumentációk készítését. A C++, C#, Java, Objective-C, Fortran és egyéb forrásokban található, speciális formátumú magyarázatok kinyerésével különböző kimenetek előállítására képes, a HTML-től a PDF-ig. Lényege, hogy a dokumentációt a kódba írjuk, ezért elég egyszerűen naprakészen tudjuk tartani. Sőt, az olvasó számára a keresztreferenciák megkönnyítik mind az olvasást, mind a megértést. Ehhez kicsit át kell alakítani kommentezési szokásokat, de a befektetés megtérül.

Speciális karakterek segítségével tudjuk megadni, hogy „észrevegye” a Doxygen generátor. Ezeket a szövegeket használja a dokumentumgenerálás során. Képes gráfokat készíteni, ezáltal grafikusan is láthatjuk az egyes részek, függvények, fájlok kapcsolatokat, hogy mi miből származik, mi mire mutat. Kiemeli és csoportosítja a kód részeit, mint például az enumerációkat, változókat, függvényeket vagy a define-okat. Továbbá az egyes dokumentációknál (pl a függvényeknél) leírja funkcióját, paramétereit, visszatérési értékét, valamint referenciáit és azt, hogy hol található a kódban. Így pontokba szedve áttekinthetővé, könnyen megérthetővé válik a kód. Minden egyes fájlra létrehoz egy ilyet, ezek között a kereszthivatkozások segítségével tudunk navigálni.

5. Összegzés

Végezetül elmondanám, hogy a házi feladatom készítése során egyértelműen kiderült, hogy a projekt- és rendszertervek segítségével könnyebb tervezni, hiszen ha előre végiggondoljuk, mit fogunk csinálni, mindig tisztában leszünk tetteinkkel. A projektterv segít az ütemezésben, pontosan látható belőle, hol tartunk a fejlesztésben, és hogy várhatóan mikor fejezzük be, esetleg mennyire vagyunk elcsúszva. A rendszertervek segítenek a tervek áttekintésében és ezek adják az első vázlatot a felépítésről. Egy jól megtervezett blokkvázlat hatalmas segítséget nyújt az építés, programozás során, hiszen teljes mértékben tisztában vagyunk vele, mit kell létrehoznunk. Ugyanebből a célból fontos a követelmény menedzsment, az egész folyamat alatt már az elejétől ismerjük az elérendő célokat és azokat a paramétereket, amivel az elkészítendő terméknek rendelkeznie kell.

Ezekhez elengedhetetlen a megfelelő dokumentáció, hiszen hiába értjük akkor, amikor éppen csináljuk, hónapokkal később nem biztos, hogy emlékszünk rá. Ráadásul egy projekten általában többen dolgoznak, ezért a hiánytalan kommunikáció biztosítása érdekében elengedhetetlen. Továbbá megkönnyíti a review-ok, prezentációk összeállítását, valamint ezeken alátámasztani a megoldások kiválasztásának okait és a miérteket. Bár néha úgy érezzük, a részletes dokumentáció fáradságos és felesleges, később gyakran rájövünk, hogy megéri. Sőt, egy projektet, vagy egy egész céget átölelő folyamatnál a pedig létfontosságú.