Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Papp Kristóf László

JÁRMŰBE ÉPÍTHETŐ VILLAMOS MEGHAJTÁSI LEHETŐSÉGEK VIZSGÁLATA

TDK dolgozat

KONZULENS

Balogh Attila

BUDAPEST, 2012

Tartalomjegyzék

Összefoglaló

Napjainkban egyre nagyobb hangsúlyt kapnak a nem fosszilis üzemanyagokkal működő járművek. Az alternatív megoldások egyik legjelentősebb képviselője a villamos meghajtás. A személygépjárművekben és néha már a kishaszongépjárművekben is megjelenő forradalmi megoldások egyre jobb hatásfokot és dinamikát tesznek lehetővé.

Jelen dolgozat célja a villamos meghajtás szempontjából releváns oldalról összehasonlítani a korszerű akkumulátortechnológiákat, motortípusokat, invertermegoldásokat, illetve az ezeket vezérlő információelektronikát. Ezek után az említett alkotóelemekből összeállításra kerül egy olyan rendszer, amely az alább felsorolt szempontok alapján optimális megoldást biztosít.

A dolgozatban részletesen tárgyalásra kerül az akkumulátor és a főáramkör méretezésének kérdése. Minden részegység szempontjából és a teljes rendszer szempontjából is a lehető legjobb hatásfok elérése volt a cél. Ezen kívül bemutatásra kerül egy, a komplett rendszert vezérelni képes információelektronika. Szoftveres szempontból összehasonlításra kerülnek a különböző vezérlő algoritmusok, ezek előnyei-hátrányai, majd bemutatásra kerül egy korszerű, térvektor-modulációra épülő algoritmus.

A tervezés szempontjai:

- A berendezés képes legyen egy 1000kg önsúlyú, 50kW-os csúcsteljesítményű gépjármű mozgatására
- A berendezéssel felszerelt gépjármű menetdinamikailag összehasonlítható legyen egy mai átlagos, benzinmotoros kisautóval
- Az akkumulátorban tárolt energiát a lehető legjobb hatásfokkal hasznosítsa az autó mozgatására
- A berendezéssel felszerelt gépjármű legalább 150km-es távot legyen képes megtenni egyetlen feltöltéssel
- A vezérlő legyen képes CAN-buszon adatokat fogadni, illetve azon keresztül információt szolgáltatni
- Feleljen meg az érintésvédelmi előírásoknak.

Abstract

Nowadays vehicles with non-fossil propulsion are getting increasingly more attention. The most important type of alternative concepts are electric vehicles (EVs). The new technologies appearing in passenger cars and sometimes even in utility vehicles are providing better and better dynamics and efficiency.

The aim of this work is to compare modern technologies in battery, electric motors, inverter circuits and control electronics with regards to their use in electric vehicles. With the help of the mentioned elements, a system is constructed, which provides an optimal solution in respect of the criteria mentioned below.

This work extensively analyses the sizing of the battery and power circuits. With regard to every component, and also to the whole system, the aim was to achieve best possible efficiency. Also shown is a possible solution for the control circuitry, which is able to fulfill every possible task in the given system. Last, the software-side solutions are also compared, and a stet-of-the art space-vector modulation scheme is presented.

Requirements posed on the system under development:

• The unit under development should be able to move a vehicle with 1000kg of curb weight with a peak power output of 50kW

• A vehicle equipped with the unit should be comparable in driving dynamics to an average modern small car

• The energy stored in the battery pack should be used with maximal efficiency to move the vehicle

• A vehicle equipped with the should be able to cover a distance of 150km with one full battery charge

• The controller should be able to receive data over the CAN-bus, and send status updates through it

• It should comply with safety regulations with regards to electric shock.

1 Bevezető

Jelen dolgozat témája a villamos hajtási lehetőségek ismertetése, összehasonlítása, illetve értékelésük járműhajtási szempontból. Villamos hajtás alatt olyan rendszert értünk, ami egy elektromos motor segítségével állít elő forgatónyomatékot. Villamos hajtásokat az élet sokféle területén találhatunk, mint pl. szállítószalagok, mozgólépcsők, felvonók. A mobil felhasználási területek közé tartoznak az elektromos járművek. Ilyenek például a villamos vasutak a tömegközlekedésből, mint a földalatti vasút, villamos, elővárosi vasút, illetve elektromos munkagépek a bányákban. Ezek a gépek a működésükhöz szükséges energiát egy előre telepített villamos hálózatból nyerik, melyre működés közben folyamatosan csatlakoztatva vannak. Léteznek azonban olyan elektromos járművek is, melyek egy meghatározott ideig képesek az elektromos hálózattól függetlenül is üzemelni, ilyenek például az elektromos autók, kerékpárok, targoncák, golfautók. A közeljövőben várható a hibridautók elterjedése, melyek a belsőégésű motor mellett egy elektromos motort is tartalmaznak, így egyesítik mindkét hajtás előnyeit.

A belsőégésű motorok magas energiasűrűségű üzemanyagok kémiai oxidációjából nyerik az energiát. Az elektromos hajtások esetében az elektromos energia szolgál „üzemanyagként”, amit azonban közvetlenül nem lehet nagy mennyiségben tárolni. Az elektromos járműveknél ezért akkumulátorokat használunk, melyek képesek a szükséges energiát kémiai úton tárolni, és azt igény esetén elektromos energiaként leadni. Az akkumulátorok mellett egyéb lehetőségek is léteznek a villamos hajtás energiával történő ellátására, ilyenek például napelemcellák, tüzelőanyagcellák vagy éppen egy belsőégésű motorral hajtott generátor. A napelemcellákról azt mondhatjuk, hogy az alacsony teljesítménysűrűségük miatt gyakorlatilag alkalmatlanok elektromos autók önálló meghajtására. Ígéretes technológia az üzemanyagcella, azonban ez a terület még mindig intenzív kutatás alatt áll. Jelenleg az elektromos autók elsöprő többségénél akkumulátorokat használnak az energiatároláshoz.

Az elektromos autók a belsőégésű motorral hajtott autókhoz képest több előnnyel is rendelkeznek. Amennyiben „tisztán” elektromos meghajtású járműről beszélünk, a lokális emissziómentesség a legfőbb előny, illetve a kőolaj alapú primerenergiahordozóktól való függetlenség. Nyilván csak akkor beszélhetünk

emisszió mentességről a teljes rendszert nézve, ha az elektromos energia előállítása során sem keletkeznek károsanyagok. Az elektromos meghajtás a kedvezőbb nyomaték-fordulatszám karakterisztikából adódóan nem igényel sebességváltót, ami a súrlódási veszteségeket csökkenti, ezzel emelve a hatásfokot. További előny, hogy üresjáratban nem igényel a meghajtás energiabefektetést. Lehetőség nyílik ezen kívül a fékezés ideje alatt az akkumulátorba energiát visszatáplálni, ami bizonyos körülmények között (hegyvidékes terep) igen jelentős energiamegtakarítást eredményezhet. A „tisztán" belsőégésű motorral hajtott járművek esetében fékezéskor a mozgási energia elveszik.

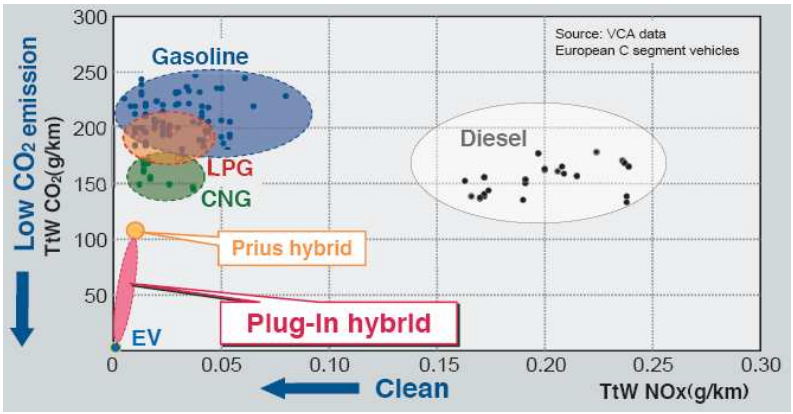
Mindezekkel együtt az elektromos hajtású gépjárművek még néhány komoly problémával kell szembenézniük. Ezek közül a legkomolyabb kihívást az energia tárolásának módja jelenti. A legmodernebb akkumulátortechnológiákkal számolva is kb. 50-szer nagyobb egy kg benzin energiasűrűsége, mint az azonos tömegű akkumulátoré. Ebből részben az következik, hogy az akkumulátortechnológia még jelentős fejlesztések előtt áll, másrészt pedig az, hogy a hajtáslánc többi elemének hatásfoka kiemelt fontosságú, hogy az eltárolt energiát minnél jobban hasznosítsuk a jármű mozgatására. Jelen dolgozat az utóbbi kérdéssel foglalkozik behatóan, hogy miként lehet a részegységekből külön-külön, és a rendszer egészéből is maximális hatásfokot kihozni.

2 A járműhajtások alapjai

2.1 Összehasonlítás a belsőégésű motor és az elektromos hajtás között

A belsőégésű motorban szénhidrogének robbanásszerű égetésével nyerünk mechanikai energiát, melyet erőkarokon, tengelyeken és fogaskerékáttételeken keresztül viszünk át a kerekekhez. Ennek a technológiának az az előnye, hogy nyers formában tudjuk felhasználni az energiahordozókat, nagy energiasűrűséggel tárolva őket. Hátránya azonban, hogy a kis méretű hőerőgépek hatásfoka relatív alacsony, ezért az energia nagy része ilyen esetben veszteségként távozik a rendszerből hő és elégetlen anyagok formájában. A nyersanyag hatalmas energiatartalmának csak kis része alakul át ténylegesen mozgási energiává, nagyobb része nem hasznosul.

Erre a problémára jelent megoltást a részleges vagy teljes villamosítása a hajtásláncnak, mivel az elektromotor hatásfoka lényegesen nagyobb mint a belsőégésű motoré. Itt azonban azzal a problémával szembesülünk, hogy nem elég a nyers energiahordozókat betölteni az autóba, hanem magasan feldolgozott szinten, villamos energiaként kell eltárolnunk. Ez problémásabb tárolást, alacsonyabb energiasűrűséget jelent. Ezen kívül nem szabad elfelejteni, hogy a villamos energia is erőművekben kerül előállításra, ahol (jelenlegi energiamix-szel kalkulálva) szintén kerül károsanyag a légkörbe, habár a hatásfok itt lényegesen jobb mint a kisebb belsőégésű motoroknál.



A CO_2 és károsanyagkibocsátás fejlődése a belsőégésű motortól a teljesen elektromos járműig [1]

2.1.1 A villamos meghajtással rendelkező gépjárművek fajtái

Micro-hybrid: Belsőégésű motoros jármű, Start-Stop funkcióval és időszakosan lekapcsolható segédberendezésekkel kiegészítve.

Mild-hybrid: Mint a Micro-hybrid, de kiegészítve fékenergia-visszanyeréssel, és elektromos rásegítéssel gyorsításnál.

Full-hybrid: Mint a Mild-hybrid, de képes tisztán elektromos hajtással közlekedni (kisebb távokon).

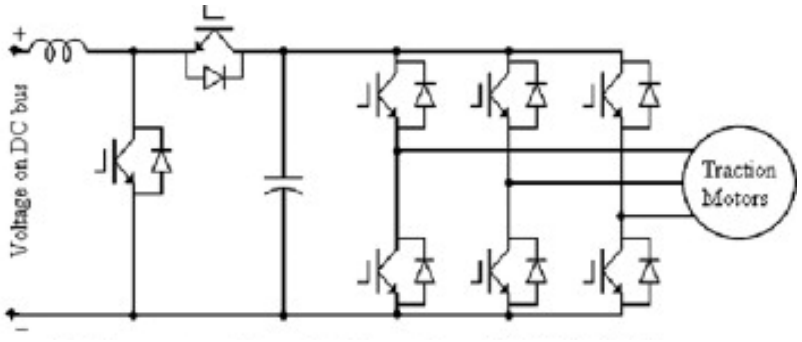
Plug-in hybrid (PHEV): Mint a Full-hybrid, de hálózatról való töltési lehetőséggel kiegészítve.

Range-extender PHEV: Itt a belsőégésű motor már minimális teljesítményre van méretezve, csak hatótáv-növelőként funkcionál.

Battery electric vehicle (BEV): Tisztán elektromos jármű, akkumulátoros táplálással.

2.2 A villamos hajtás felépítése

A villamos hajtás legfontosabb elemei az akkumulátor, a DC/DC konverter (opcionális), az inverter és a motor. Az akkumulátor szolgáltatja a működéshez szükséges energiát, amit a DC/DC konverter alakít át szükség esetén magasabb feszültségre, ezt követően pedig a háromfázisú inverter táplálja a villanymotort. A villanymotor általában sebességváltó nélkül, egy fix áttétellel hajtja a kerekeket.



A villamos hajtás felépítése [7]

3 A villamos hajtás elemeinek kiválasztása

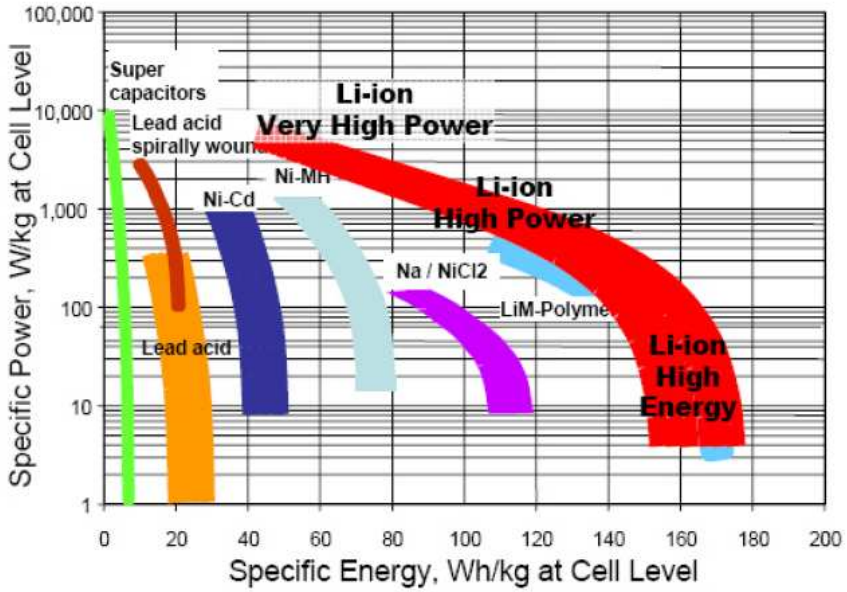
Ebben a fejezetben részletesen megvizsgálom a villamos hajtás minden elemét, összehasonlítom a rendelkezésre álló technológiákat, és javaslatot teszek villamos járművekben történő alkalmazásukra.

3.1 Akkumulátor választása

Az akkumulátor a villamos hajtású járművek alapvető energiaforrása. Tulajdonsága, hogy kémiai energia formájában tárolja a villamos energiát. Alapvetően kijelenthető, hogy jelenleg az elektromos autók elterjedésének legfőbb korlátja az akkumulátortechnológia fejlettségi szintje, pontosabban az, hogy minden típusnak jelenleg lényegesen alacsonyabb a fajlagos energiasűrűsége, mint a folyékony tüzelőanyagoké. Ennek eredményeképp azonos hatótáv megtételéhez lényegesen nagyobb tömegű akkumulátorcsomagot kell beépíteni a járműbe, mint amennyi folyékony tüzelőanyagból szükséges volna, ezáltal az egész jármű tömege megnő, ami negatívan hat a menetdinamikára és a helykínálatra. Ezt a tényt valamennyire kompenzálja a villamos hajtáslánc belsőégésűnél magasabb hatásfoka, azonban a különbség még mindig jelentősnek mondható. Továbbá megemlítendő, hogy az akkumulátor a nagyobb hatótávú elektromos konstrukciók (PHEV-től fölfelé) legdrágább alkatrésze, mely lényegesen befolyásolja a gépjármű árát is.

Ezen okokból kifolyólag megállapítható, hogy az elkövetkezendő években az akkumulátortechnológia fejlesztésére fog a legnagyobb figyelem irányulni, mivel ez az alkatrész döntően befolyásolja a jármű piacképességét. Szintén emiatt kifejezetten fontos az akkumulátortípus helyes megválasztása tervezésnél is.

A következő ábrán láthatóak a manapság elérhető villamosenergia-tárolási technológiák fajlagos energia- és teljesítménysűrűség szempontjából. Látható, hogy jelenleg a Li-polimer illetve Li-ion típusú akkumulátorok biztosítják a legmagasabb fajlagos energiasűrűséget, mely az elektromos autók szempontjából kiemelkedően fontos. A Li-ion típusú akkumulátorok további előnye, hogy nem jellemző rájuk a memóriaeffektus. Li-ion akkumulátorcsaládba többféle technológia is tartozik, melyeket a következő alpontban ismertetek.



A különböző energiatárolók fajlagos energia- és teljesítménysűrűsége [2]

3.1.1 A lítium-ion típusú akkumulátorok

Lítium-ion típusú akkumulátorokat először az 1970-es években javasoltak az Exxon mérnökei. Az első működő prototípus lítium-kobalt-oxid ($LiCoO_2$) elektródákkal 1979-ben készült el 4V-os cellafeszültséggel. A lítium-ion típusú akkumulátorok manapság a legelterjedtebben használt akkumulátortípusok a szórakoztatóelektronikában. Magas fajlagos energiasűrűség és teljesítménysűrűség jellemzi őket, ezen kívül több előnyös tulajdonsággal is rendelkeznek, mint például a memóriaeffektus hiánya, illetve az alacsony önkisülés. Memóriaeffektus alatt azt a jelenséget értjük, amikor az akkumulátor visszatérő, részleges kisütése esetén, lassan veszít a kapacitásából, ez elsősorban a NiCd (nikkel-kadmium) akkumulátoroknál fordul elő. Ebből az okból kifolyólag a NiCd akkumulátorokat nem szokás villamos járművekben alkalmazni, mivel ezeknél jellemzően nem merítjük le teljesen az akkumulátorokat. A lítium-ion akkumulátorok napjainkban intenzív kutatás alatt állnak, hogy hatékonyabban alkalmazhatóak legyenek a villamos járművekben is. A lítium-ion típusú akkumulátorok családjába többféle felépítéssel rendelkező modell tartozik, az anód és a katód anyagától függően, ezek tulajdonságai némiképp eltérőek. A fontosabb felépítéseket az alábbiakban ismertetem.

Lítium-kobalt-oxid ($LiCoO_2$) katóddal rendelkező cella: Ez a legrégebbi típus, a fajlagos energiasűrűsége jónak mondható, viszont előállítási költsége magas, és erősen robbanásveszélyes bizonyos helyzetekben (túltöltés vagy fizikai behatás esetén).

Szórakoztatóelektronikai készülékekben gyakran használatos, azonban villamos járművekbe túl veszélyesnek mondható.

Lítium-mangán-oxid ($LiMn_2O_4$) katóddal rendelkező cella: Ez a típus is széles körben elterjedt, alacsony előállítási költség, azonban rövid élettartam és viszonylag alacsony fajlagos energiasűrűség jellemzik. Villamos járművekben történő alkalmazása főleg ez utóbbi miatt nem előnyös.

Lítium-vas-foszfát ($LiFePO_4$) katóddal rendelkező cella: A típus legfőbb előnyei közé tartozik a nagyfokú stabilitás (alacsony robbanásveszély fizikai behatás illetve túltöltés esetén, a többi típushoz képest) és a relatív magas fajlagos energiasűrűség. További előny a széles tartományban gyakorlatilag állandó cellafeszültség. Ez a típus mondható a legrobosztusabbnak, képes elviselni kiugróan magas kisütő-áramokat is. Tulajdonságai alapján ez a típus mondható a legideálisabb választásnak villamos járművek energiatárolójaként történő alkalmazásra.

Bizonyos lítium-polimer cellák magasabb fajlagos energiasűrűséget is elérhetnek, mint a lítium-ion típusúak, azonban ezek lényegesen robbanásveszélyesebbek és rövidebb élettartalmúak, ezért villamos járművekben nem célszerű használatuk. A lítium-polimer technológia lényege, hogy az elektrolit nem egy szerves oldószerben „úszik", hanem valamilyen polimerben található megkötve. [3]

Ha egy 1000kg önsúlyú kisautót veszünk, vezetési stílustól függően kb. 16kWh kapacitású $LiFePO_4$ akkumulátorra van szükség 100km megtételéhez, így a továbbiakban ezzel a kapacitással számolok.

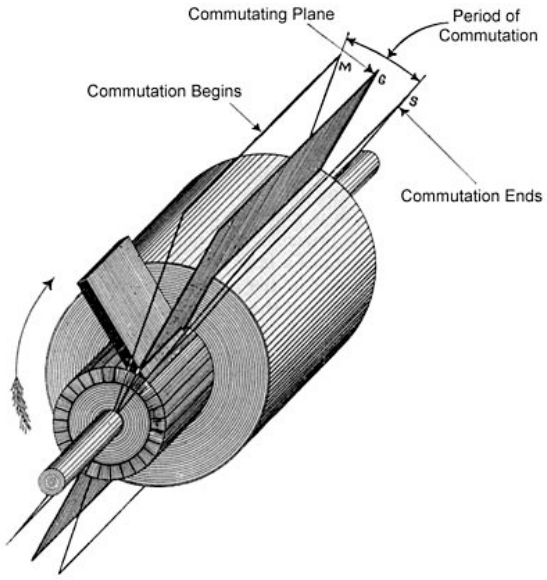
3.2 Motortípus választása

A villamos hajtás alapvető eleme a villanymotor. A motor típusának helyes kiválasztása a legfontosabb lépés a hajtás további elemeinek megtervezése előtt. A motor feladata az inverter által rendelkezésre bocsátott elektromos energiát mozgási energiává alakítani. A motor kiválasztásánál figyelembe kell venni annak hatásfokát, előállítási költségét, méreteit, súlyát, a szükséges kiegészítő elektronikát.

A villanymotorok működési elve egységesen a mágneses mezők kölcsönhatásán alapul. Az elvi hátteret megadó törvényszerűségeket Michael Faraday fedezte fel 1831-ben, az indukciós törvény részeként. A rotor és az állórész mágneses mezői erőket hoznak létre, melyek a rotort elmozdulásra késztetik. A rotor és az állórész is

tartalmazhat állandómágnest, tekercset, ferromágneses anyagot (vasat) vagy vezetőt (rezet), ezek kombinációja adja meg, hogy milyen típusú a motor. A főbb típusokat az alábbiakban ismertetem.

Egyenáramú motor. Ennél a motortípusnál a forgórész tekercselést tartalmaz, az állórész pedig állandómágnest vagy tekercselést. Az állórész egy állandó mágneses teret hoz létre, amely lehet változó erősségű, ha tekercselt állórészt használunk. A forgórész egy erre merőleges mágneses teret hoz létre, így maximalizálva az elérhető nyomatékot. Mivel a forgórész folyamatos mozgásban van, a tekercseket egy kommutátoron keresztül kell gerjeszteni, mely biztosítja, hogy mindig az állórész mágneses terére merőleges tekercs legyen aktív. A kommutátor egyben az egyenáramú motor leggyengébb alkatrésze is, mivel folyamatosan nagy áramokat kell indítania és megszakítania. A kommutátor általában szénkefés felépítésű, a tekercsek kivezetéseit álló szénkeféken keresztül kapcsoljuk a motor bemenő kapcsaira. További hátránya a kommutátor alkalmazásának, hogy a folyamatos kefeszikrázás miatt nem alkalmazhatóak az ilyen motorok robbanásveszélyes térben, illetve a kommutátor plusz alkatrészként megnöveli a motor méreteit. Mivel a szénkefék a fizikai igénybevétel miatt folyamatosan kopnak, a kommutátor cseréje rendszeres időközönként szükséges. Az egyenáramú motornál gyakorlatilag egyetlen előnyként az egyszerű szabályozhatóságot említhetjük, a motor nyomatéka egyenesen arányos a kapcsain átfolyó árammal, a fordulatszáma pedig a kapocsfeszültséggel. Ilyen módon egy egyszerű egyfázisú egyenfeszültség elegendő a kapcsain, melyet könnyen lehet előtét-ellenállással vagy impulzusszélesség-modulált vezérléssel állítani. Korábban, a gyors teljesítményelektronikai alkatrészek megjelenése előtt egyedül ez a motortípus volt könnyen szabályozható, azonban manapság a hátrányai miatt elavultnak tekinthető.

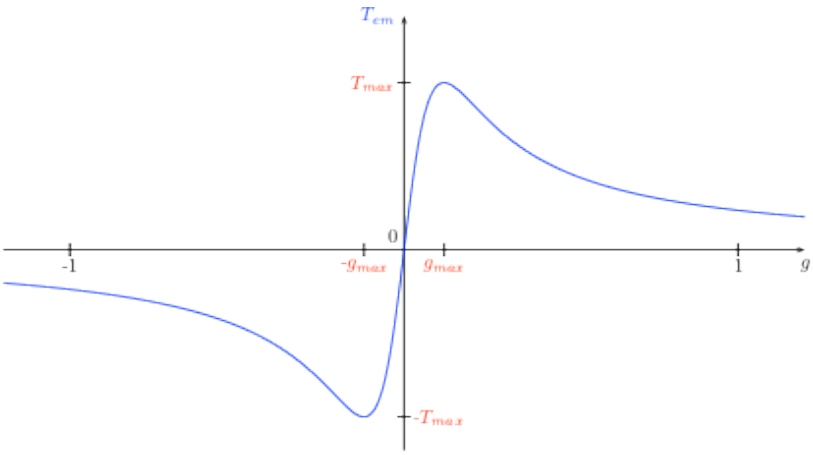


A kommutátor működése [5]

Aszinkronmotor. Szokás indukciós motornak is nevezni, tekercselt állórészt és egyszerű vezetőt (rezet vagy alumíniumot) tartalmazó forgórészt tartalmaz. Az állórész tekercseire kapcsolt váltakozó áram változó mágneses tere feszültséget indukál a forgórészben ez a feszültség pont olyan áramot hoz létre, amelynek mágneses tere a létrehozó mágneses tér ellen hat, ezáltal nyomaték keletkezik az állórész és forgórész között. A nyomaték mértéke a szlip-től függ.

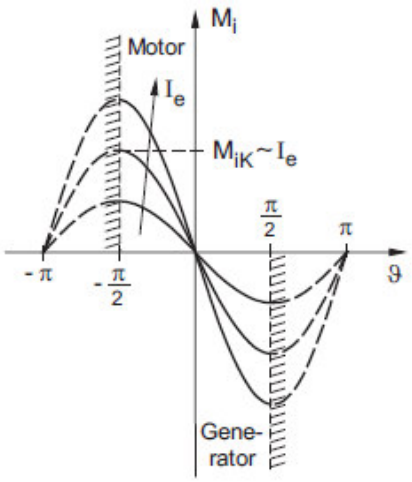
$$s = \frac{n_s - n_r}{n_s}$$

A szlip álló rotornál 1, szinkron fordulatszámon 0. Szinkron fordulatszámon nem tud a motor nyomatékot kifejteni, mivel nincs indukált feszültség, ebből következik, hogy a motor terhelt állapotban nem tud szinkron fordulatszámon üzemelni, mindig szükség van egy kis szlip-re. Felépítése egyszerű, a forgórész általában ketreces kivitelű, egyszerű réz vagy alumínium vezetőkkel. Az állórész általában háromfázisú tekercselésű. Az egyszerű forgórészfelépítésnek köszönhetően a motor tehetetlenségi nyomatéka nagyon alacsony. Ezzel együtt ez mondható a legrobosztusabb motortípusnak, egyszerűségéből adódóan nagyon igénytelen, karbantartásmentes. Méreteit tekintve kompakt, kis súlyú. Hátránya, hogy háromfázisú, frekvenciaváltós táplálást igényel, illetve hogy nem képes szinkronfordulatszámon üzemelni, ezáltal még állandó terhelésnél is zártláncú fordulatszámszabályzást igényel. Mindent összevetve az aszinkronmotor igen jól használható villamos járművekben.



Az aszinkronmotor nyomatéka a szlip függvényében [6]

Szinkronmotor. Szokás még BLDC (brushless DC) motornak is nevezni, állandómágneses vagy tekercselt forgórészt tartalmaz és tekercselt állórészt. A forgórész feladata csak egy állandó mágneses tér előállítása, ezért tekercselt forgórész esetén is elegendő a csúszógyűrűk alkalmazása, nincs szükség kommutátorra. Az állórész tekercselése általában háromfázisú, amely a változó mágneses tér előállításáért felelős. A motor csak szinkronfordulatszámon képes üzemelni, azaz a rotor az állórész által előállított mezőt maradéktalanul követi, a nyomaték a kettő között létrejövő fázisszög függvénye. Inverteres táplálásnál úgy szokás beállítani a fázisszöget, hogy felbontjuk két komponensre, d-re (direct, közvetlen), és q-ra (quadrature, merőleges), majd megpróbáljuk az áram d komponensét 0-n tartani, mivel ez a komponens nem hoz létre nyomatékot, csak a tekercseken okoz veszteséget. A q komponens változtatásával pedig lehetőség van nyomatékot adni a motorra, mivel ez a 90°-nak megfelelő fázisszög.

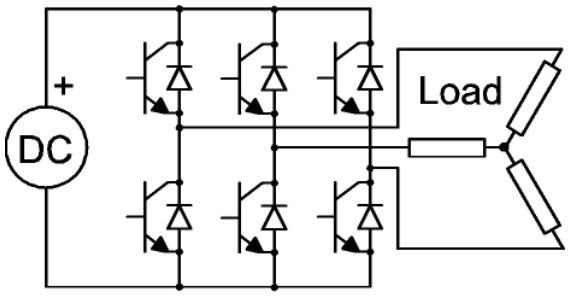


A szinkrongép fázisszög- nyomaték karakterisztikája

Tekercselt állórészt nagyteljesítményű motoroknál és generátoroknál szokás alkalmazni, mivel a gerjesztés a nyomaték beállításának másik egyszerű lehetősége. A szinkronmotor szintén kompakt felépítésű, bár a bonyolultabb forgórész miatt nem annyira könnyű, mint az aszinkronmotor. Ezzel együtt az elérhető nyomaték is magasabb. Az egyenáramú motorral ellentétben itt is háromfázisú frekvenciaváltós vezérlésre van szükség a működtetéséhez. Előállítási költsége magasabb az aszinkronmotorénál. Összegezve a tulajdonságokat, ez a típus is igen alkalmas villamos járművekben való alkalmazásra, az egyszerű felépítésnek és a nagy nyomatéknak köszönhetően a továbbiakban ezzel a motortípussal foglalkozok. Szabályzás szempontjából (amennyiben fordulatszám-jeladót használunk) nincs lényeges különbség az aszinkronmotorhoz képest.

3.3 Inverter megtervezése

Ahhoz, hogy a háromfázisú tekercseléssel ellátott motort vezérelni tudjuk, szükség van egy frekvenciaváltóra, amely képes a szükséges feszültséget és frekvenciát előállítani. Az inverter felépítését tekintve egy háromfázisú hídkapcsolásból áll, melyet egy digitális vezérlőelektronika hajt meg, a mért adatok alapján.

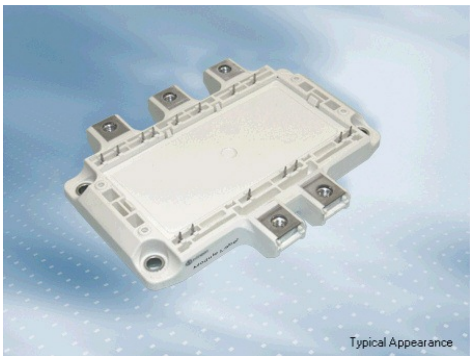


Háromfázisú hídkapcsolás [4]

A hídkapcsolás hat darab teljesítményelektronikai kapcsolóból áll, melyekkel párhuzamosan visszafele van kapcsolva egy dióda, mely megengedi az áramok záródását kikapcsoláskor. A teljesítményelektronika lehet MOSFET vagy IGBT. Bár a MOSFET-ek előnyösebb tulajdonságokkal rendelkeznek a bekapcsolási-kikapcsolási időket tekintve, nem képesek ellátni a villamos meghajtás teljesítményigényét. 550V-os buszfeszültséggel számolva kb. 91A áramerősségre van szükség a 50kW-os csúcsteljesítmény eléréséhez. Ez jelenleg csak IGBT-kkel elérhető.

Manapság a háromfázisú hajtások igényeire gondolva gyártásba kerültek olyan IGBT modulok, melyek a hat tranzisztort és diódát egy tokozásban tartalmazzák, ezzel segítve a hűtés megtervezését, és a kompakt méret kialakítását. Ennek a megoldásnak további előnye, hogy az alkatrészek egymáshoz vannak válogatva, nem kell félni, hogy egyik szignifikánsan más paraméterű lesz majd mint a másik. Jelenleg a Fuji Electric, a Powerex és az Infineon számítanak ilyen modulok nagy gyártóinak.

550V DC feszültséggel és 91A csúcsárammal kalkulálva ideális választás az Infineon FS100R07PE4 modul, mely egyben tartalmazza a három hídágat és egy hőmérő NTC termisztort. Maximális kollektor-emitter feszültsége 550V, maximális kollektorárama 100A.



Infineon FS100R07PE4 IGBT modul

A teljesítményelektronika azonban nem tud közvetlenül a mikrokontrollerhez csatlakozni, az IGBT-k meghajtásához szükség van egy meghajtó (driver) áramkörre, mely a gate-eket feltölti, illetve kisüti. Ehhez tipikusan +15V, -8V feszültséget használnak. A kapcsolók gyors be- és kikapcsolása a veszteségek minimalizálását jelenti, mivel átmeneti állapotban keletkezik a legnagyobb veszteség a kapcsolókon. A driver lehet saját tervezésű áramkör, mely egy leválasztott +15V, -8V-os tápforrást alkalmaz, vagy előre legyártott. Jelen esetben az egyszerűség és megbízhatóság okán egy előre gyártott megoldás mellett döntöttem.

A CT Concept cég terméke a 2SC0108T2A0 driver, mely 1700V feszültségig tud üzemelni, és belül állítja elő a -8V-os tápfeszültséget. 8A maximális áramot képes szolgáltatni az IGBT-k gate-jeinek, és két csatornát tartalmaz, így három darab kell belőle a három hídág ellátására. Ezen kívül tartalmaz beépített túláramvédelmet minden csatornán.

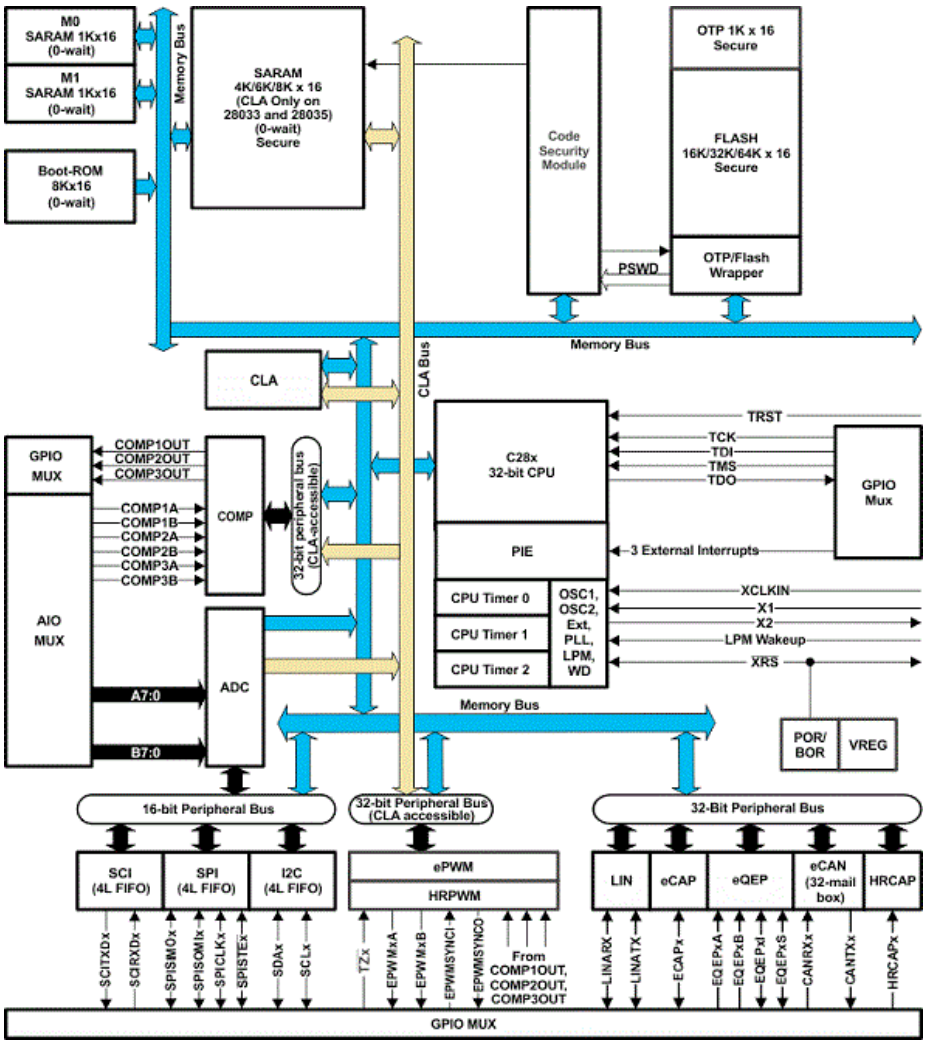
CT Concept 2SC0108T2A0 driver

3.4 Processzor kiválasztása

A teljesítményelektronika vezérléséhez, illetve a teljes rendszer szabályzásához elengedhetetlen valamilyen információelektronika használata. Ennek egyik oldalról elegendően gyorsnak kell lennie ahhoz, hogy a szükséges számítási feladatokat a kívánt időközönként elvégezze, másrészt viszont költséghatékony és egyszerűen programozható processzort kell választani. Általánosságban elmondhatjuk, hogy ahhoz, hogy a PID szabályzókat 20kHz-es gyakorisággal futtatni tudjuk és a szükséges kommunikációt (pl. CAN) is el tudjuk végezni, minimum 32bites mikrokontrollerre van szükség, ez alatti bitszámok esetén már kritikus lehet az időszeletek felosztása.

Itt lehetőség van többféle architektúra választására is, jelen munkában én a Texas Instruments cég **TMS320F28035 Piccolo** típusú DSP-je mellett döntöttem, mely kifejezetten motorvezérlési célokra készült. Ez a processzor 60MHz-es órajellel fut és 128kB Flash memóriát tartalmaz, ami a tapasztalatok alapján elegendő minden szükséges funkció megvalósítására. Az, hogy ez a processzor kifejezetten motorok szabályzására készült, abban nyilvánul meg, hogy egy sor olyan perifériát tartalmaz, melyek megkönnyítik az ilyen jellegű rendszerek felépítését.

Ezek közül legfontosabb a *CLA*, azaz a Control Law Accelerator. Ez a modul lényegében egy második 32 bites lebegőpontos processzor, mely a főprocesszortól függetlenül, de ugyanazon az órajelen fut, és a legfontosabb perifériákhoz közvetlenül hozzfér. Ezzel lehetővé válik például, hogy az ADC érték beolvasása után lefusson egy áramszabályzó algoritmus, és rögtön beállíthassuk a PWM csatonák kimenetét, miközben a főprocesszor a kommunikációval, illetve magasabb szintű szabályzással foglalkozhat.



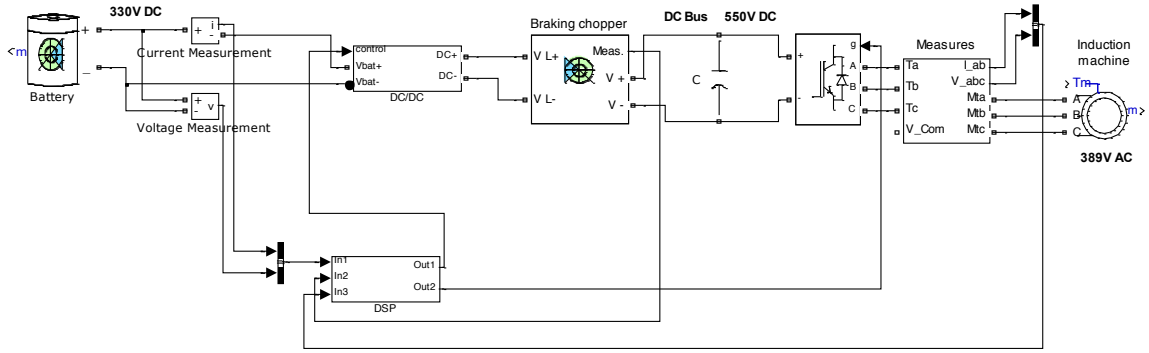
A TMS320F28035 belső felépítése [8]

További jelentős periféra az ePWM (Enhanced PWM), mely képes egyetlen számláló alapján komplementer kimeneteket képezni a hídágakhoz, és beállítható holtidővel (dead-band) rendelkezik. Ezen kívül megtalálható egy inkrementális adó fogadó modul (eQEP), valamint az autós alkalmazásokban elengedhetetlen CAN-busz interfész.

3.5 Rendszerterv

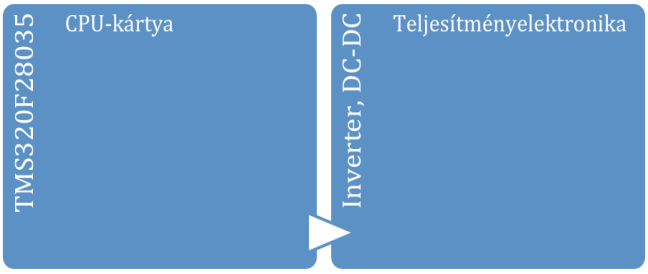
A teljes rendszer áll tehát az akkumulátorból, melynek kapcsaira a DC/DC konverter kapcsolódik az 550V-os buszfeszültség előállítására. Lehetőség van DC/DC konverter nélkül is üzemeltetni a rendszert, illetve magasabb akkumulátorfeszültséget beépíteni, azonban ez biztonsági szempontokból nem előnyös, a motor jobb kihasználtsága érdekében viszont érdemes a magasabb DC-buszfeszültséget használni.

A DC-buszra kapcsolódik a fékchopper, mely a túlfeszültség ellen védi a rendszert.



A teljes rendszer modellje MATLB Simulink-ben

A megvalósítás két áramkörre épít, hogy a nagyfeszültséget kellően leválasszuk a kommunikációs csatárnáktól. A CPU-kártya és a teljesítményelektronikát tartalmazó kártya között kábel biztosítja az összeköttetést.



A két kártyás megvalósítás

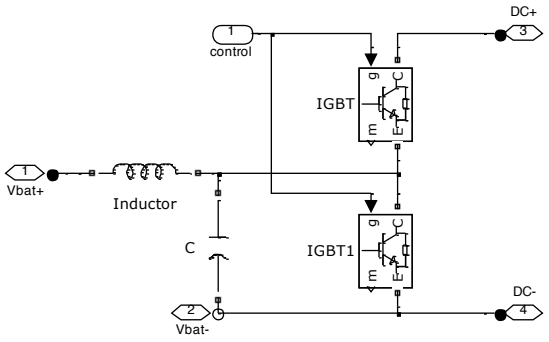
3.5.1 Visszatápláló fékezés

Felmerül a kérdés, hogy lehetséges-e egy autót kizárólag elektromos úton fékezni. Ha egy 1000kg önsúlyú gépjárműből indulunk ki, amely 100km/h sebességről lassít állóhelyzetig 40m alatt, azt mondhatjuk, hogy átlagosan kb. 111kW fékteljesítménnyel kell számolnunk. A LiFePO4 típusú akkumulátorok maximum kb. 1C-vel, azaz a kapacitásának megfelelő árammal tölthetőek, ami 16kWh esetén csak 16kW-ot jelent. A fennmaradó teljesítményt a fékellenállással kellene eldisszipálni, azonban ez alig kivitelezhető az ellenállás méretéből és hűtésigényéből fakadóan.

Láthatjuk tehát, hogy bár a visszatápláló fékezés hasznos a jármű energetikáját illetően, nem helyettesíti az üzemi féket!

3.6 Akkumulátormenedzsment

A lítium-ion típusú akkumulátoroknál kiemelkedően fontos az akkumulátormenedzsment helyes kivitelezése. Ezért figyelni kell mind az akkumulátor kapocsfeszültségét, mind az áramát, amivel szabályozni tudjuk a visszatápláló fékezés esetén képzett töltési fázist. A töltést és a kisütést a már említett DC/DC konverter végzi, amely egy hagyományos buck-boost koverter két IGBT-vel. Az IGBT-k vezérlését szintén a DSP végzi.



A DC/DC konverter modellje

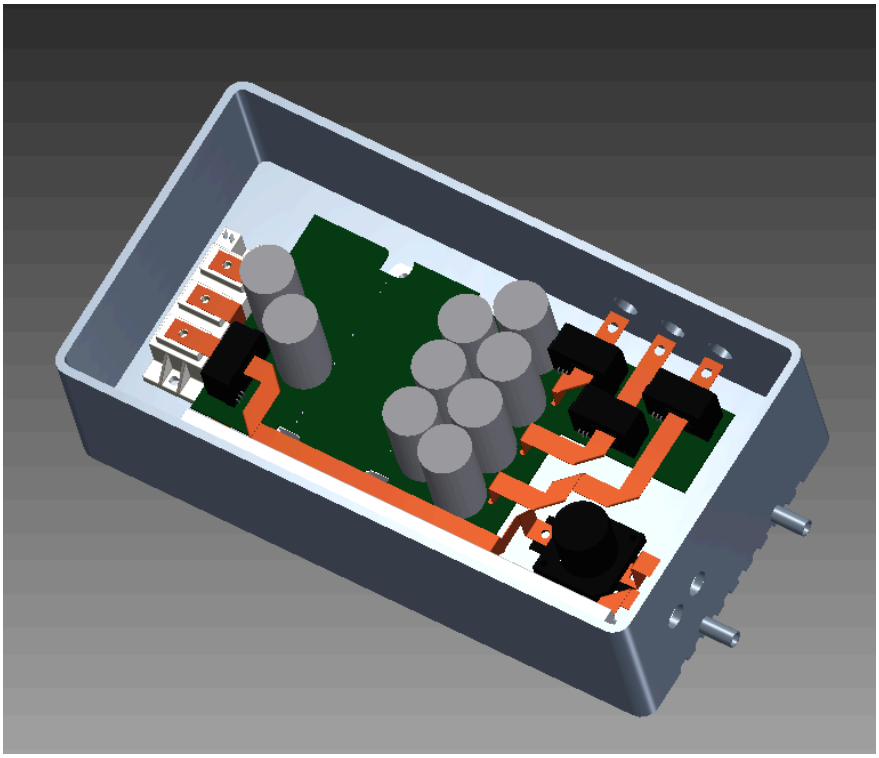
3.7 Hűtés megtervezése

Mivel mind az inverter IGBT modulja, mind a DC/DC konverter két IGBT-je jelentős hőveszteséggel bír, ezek hűtésére külön figyelmet kell fordítani. Hogy maga az inverter kompakt felépítésű legyen, ezeket az alkatrészeket egy házba célszerű elhelyezni, mely egyben a hűtést is elvégzi.

Egy lehetséges kialakítás, ha egy vízhűtéssel egybeépített alumíniumházat használunk, és ebbe helyezzük el a nagy hőleadással rendelkező alkatrészeket. Az Autodesk Inventor szoftver segítségével készítettem egy tervet a lehetséges elrendezésre, mely az alábbiakban látható.

A vízhűtésen kívül szükség van még a házban egy ventilátorra is, mivel a panelen egyéb kisebb alkatrészek is adnak le hőt, és összességében a levegő is túlhevülhet a házban ha nem keringetjük. Mind a vízhűtés mind a léghűtés vezérlés a DSP végzi.

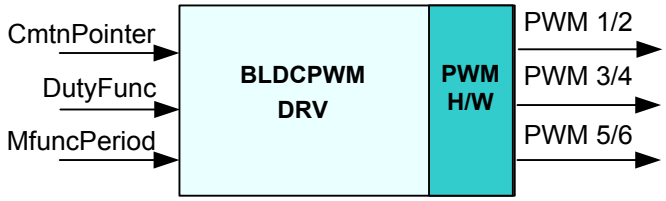
Az inverter tervezett háza



Az inverter tervezett háza

4 A vezérlő szoftver felépítése

A vezérlő szoftver elkészítéséhez segítséget nyújt a Texas Instruments ControlSuite könyvtára, mely ingyenesen letölthető a gyártó honlapjáról (http://www.ti.com/tool/controlsuite). A könyvtár többféle processzorcsaládhoz használható, többek között a TMS320F2803x családhoz is. Ennek a könyvtárnak része a DMC (digital motor control) csomag, amivel részletesebben foglalkozni fogok. A DMC könyvtár C-függvényekből és makrókból áll, melyek a hajtásszabályzás valamelyik részfeladatát látják el. Ezzel az egységes rendszerrel elérhető, hogy a szoftver egyszerre legyen moduláris és alkalmazásfüggetlen.



A BLDCPWM modul

Mivel a modulok interfésze egységes, egyszerűen egymáshoz kapcsolhatóak. Példaként a motorvezérlő (BLDCPWM modul) inicializálása:

```
PWMGEN pwm1 = PWMGEN_DEFAULTS;

main()
{
   pwm1.PeriodMax = 1000;
   BLDCPWM_INIT_MACRO(pwm1);
}

void interrupt periodic_interrupt_isr()
{
   pwm1.CmtnPointer = (int)(CmtnPointer1);
   pwm1.DutyFunc = (int)_IQtoIQ15(DutyFunc1);
   BLDCPWM_MACRO(pwm1);
}
```

A DC-buszfeszültség szabályzóját a CLA segítségével implementáltam, mivel így ezt a terhet levehetjük a CPU-ról, és ez a feladat nagyrészt független a többi szabályzási feladatról. A CLA programozható C illetve Assembly nyelven is. Ezúttal Assembly nyelven valósítottam meg az algoritmust a gyorsabb működés érdekében:

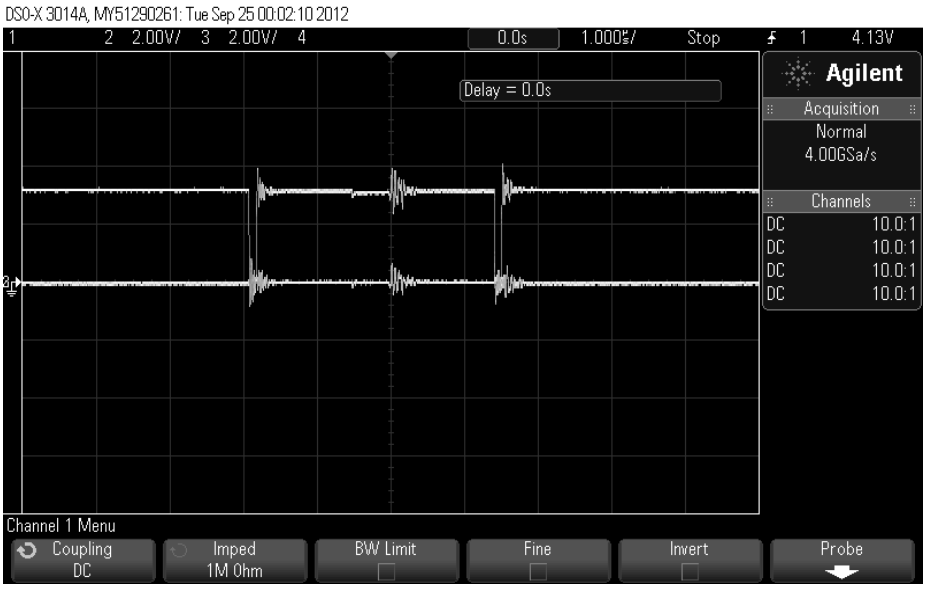
```
_DCBooster:
    MMOVZ16     MR0, @_DCBooster_enable
    MMOVZ16     MR1, @_Power_backflow
    MMOVIZ      MR2, #0x0000
    MMOVXI      MR2, #0x0001
    MXOR32      MR1, MR1, MR2
    MAND32      MR0, MR0, MR1
    MNOP
    MNOP
    MNOP
    MBCNDD      _DCBooster_off, EQ
    MNOP
    MNOP
    MNOP

    MMOV32      MR0, @_ADC5_filtered
    MMOV32      MR1, @_DCBooster_target
    MSUBF32     MR1, MR1, MR0
    MMOV32      MR0, @_DCBooster_istate
    MADDF32     MR0, MR0, MR1
    MMOV32      @_DCBooster_istate, MR0
    MMOV32      MR2, @_DCBooster_pgain
    MMPYF32     MR1, MR1, MR2
    MMOV32      MR2, @_DCBooster_igain
    MMPYF32     MR0, MR0, MR2
    MADDF32     MR2, MR0, MR1
    MMAXF32     MR2, #0.0
    MMINF32     MR2, #500.0
    MSUBF32     MR2, #1000.0, MR2
    MF32TOUI16R MR3, MR2
    MMOV16      @_EPwm2Regs.CMPB, MR3
    MMOVIZ      MR0, #0x0000
    MMOVXI      MR0, #1000
    MMOV16      @_EPwm2Regs.CMPA.half.CMPA, MR0
```

5 Mérések

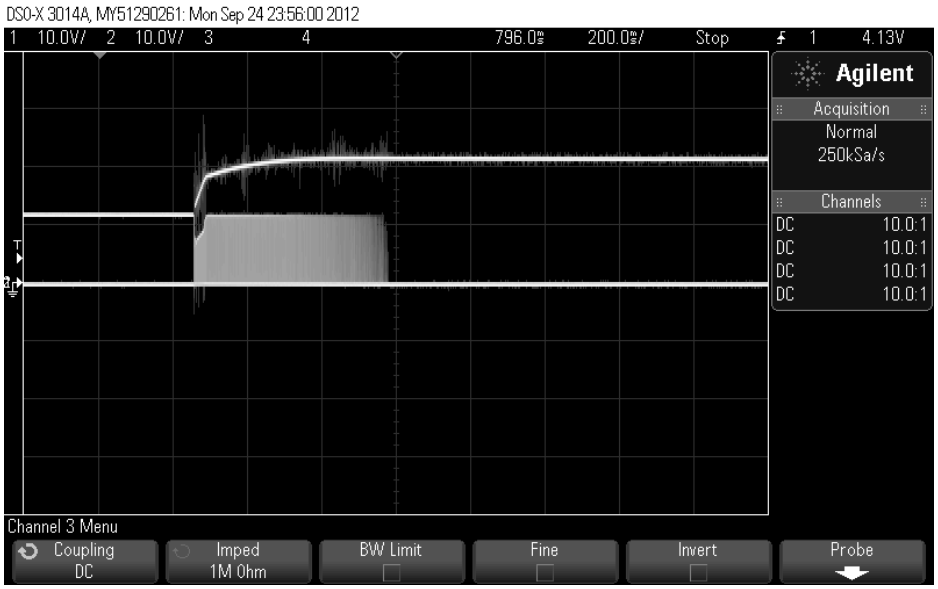
A tervezett eszközből készült egy leméretezett prototípus, amellyel költséghatékony módon lehet vizsgálni a szabályzási algoritmusok működését, illetve a kommunikációt. A prototípus legfeljebb 500W kimenőteljesítmény leadására képes, ami már elegendő ahhoz, hogy a hajtást minden szempontból megvizsgáljuk egy kis méretű szinkronmotor használata mellett. A prototípus kapcsolási rajza a függelékben található.

Mivel az IGBT-knek szükségük vanbizonyos időre ahhoz, hogy teljesen bekapcsolt állapotból teljesen kikapcsoljanak, illetve fordítva, a hídágaknál vigyázni kell, nehogy az átmeneti tartományban zárlatot okozzunk a DC-buszon. Ennek elkerülésére a kontroller beépített dead-band funkcióját használtam, melyet 0.1us holtidőre állítva már biztonságosan elkülönül a bekpcsolási és kikapcsolási folyamat.



A 0.1us holtidő (dead-band) megjelenése

A DC-buszfeszültség előállítására egy PI szabályzót alkalmaztam, melyet 20kHz-es mintavételi frekvenciával futtatva 20V-ra szabályozom be a DC-buszfeszültséget. A szabályzó hangolása először terheletlen állapotban történt, ugyanis itt a legnagyobb a veszélye annak, hogy oszcillálni kezd a rendszer. Az erősítések folyamatos emelésével a P-tag erősítését 2.0-ra, az I-tagét 0.0005-re állítottam be, ez bizonyult a stabilitás határának. A következő ábrán látható a szabályzó működés közben, amikor 12V-ról 20V-ra szabályozzuk a feszültséget terheletlen állapotban.



A DC-buszfeszültség szabályzója működés közben

6 Összefoglalás, értékelés

Általánosságban elmondhatjuk, hogy az alternatív járműhajtások, az elektromobilitás egyre fontosabb szerepet kapnak a járműgyártásban. A folyamatosan növekvő nyersanyagárak, az egyre szigorodó szabályzások a károsanyagkibocsátásra a villamos járműhajtások felé irányítják a figyelmet. Jelen dolgozatban megpróbáltam összefoglalni egy villamos jármű legfontosabb tervezési szempontjait, majd minden egyes komponensre külön lebontva, és együtt, rendszerszinten is megvizsgálni a leghatékonyabb megoldásokat.

Összességében elmondható, hogy jelenleg a villamos járműhajtások széleskörű elterjedésének legnagyobb gátja az akkumulátortechnológia ára. Amíg az elektromos hajtású autó a belsőégésű motorral szerelt járművekkel versenyképes hatótávval nem szállítható versenyképes áron, nem várható a technológa áttörése. Megállapítható azonban, hogy több technológiai korlátja nincsen rendszernek, azaz ha megoldódik az energiatárolás problémája, semmi nem áll a széleskörű elterjedés útjában, minden egyéb probléma megoldottnak tekinthető, ahogy arra a dolgozatban is rávilágítok.

Irodalomjegyzék

[1] Abe S. 2008: Toyota Hybrid Systems – Review of 10 years of mass production, aus Proceedings: Engine & Environment 2008: 120g CO2/km – what about driving fun and costs, 20th international AVL Conference “Engine & Environment”, Graz, September 2008.

[2] Köhler Uwe 2007: Li-Ion battery systems for the next generation of hybrid electric vehicles, in 19th International AVL Conference “Engine & Environment”, September 6th - 7th, Graz, Austria, 2007

[3] Wikipedia: *Lithium-ion battery*, http://en.wikipedia.org/wiki/Lithium-ion_battery

[4] Wikipedia: http://en.wikipedia.org/wiki/File:3-phase_inverter_cjc.png

[5] Wikipedia: *Brushed DC electric motor,* http://en.wikipedia.org/wiki/Brushed_DC_motor

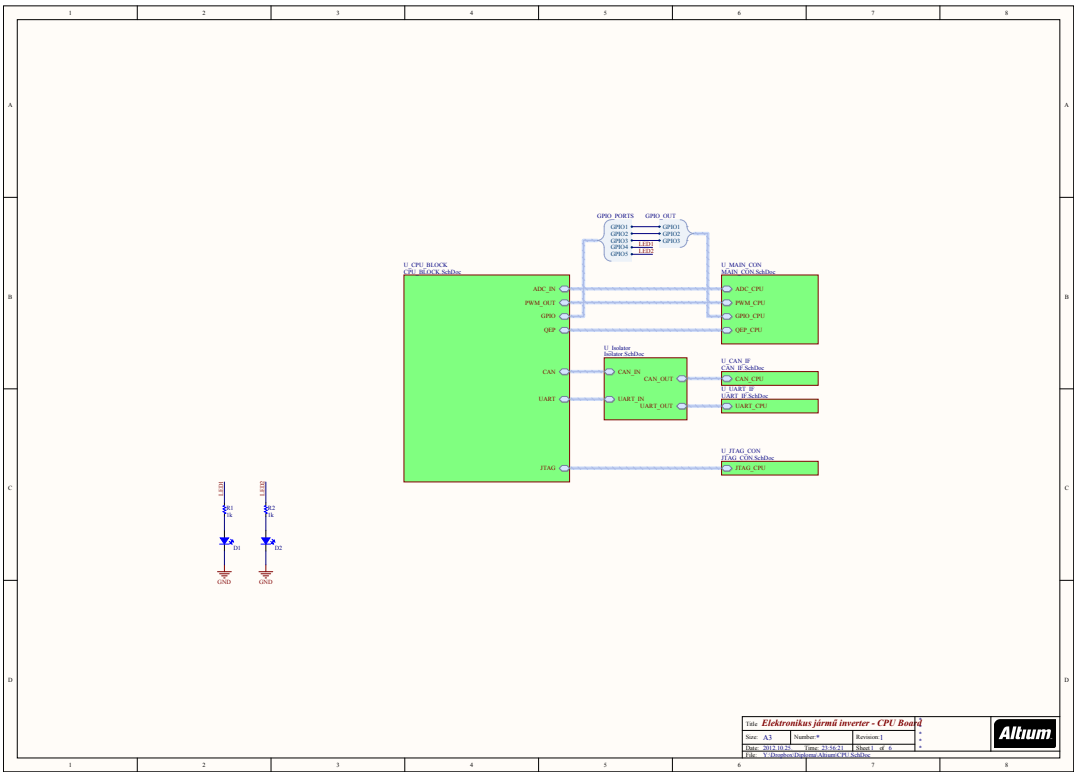
[6] Wikipedia: *Induction motor,* http://en.wikipedia.org/wiki/Induction_motor

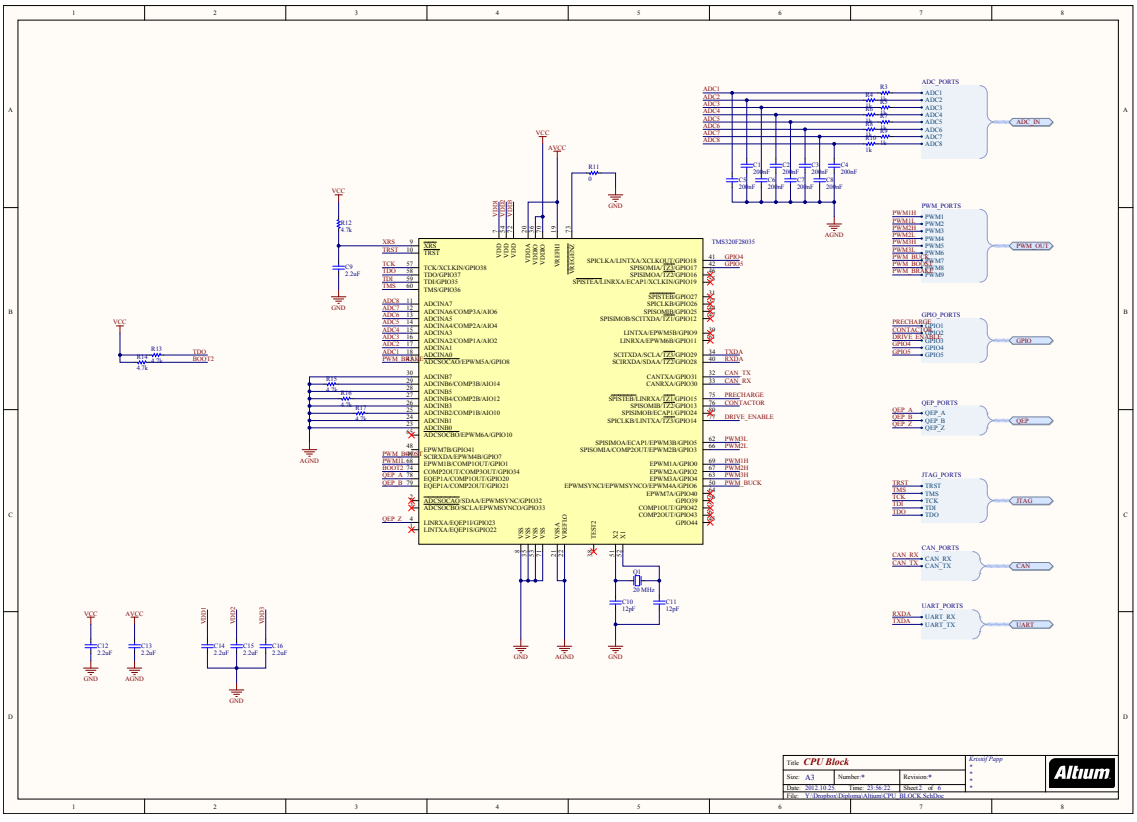
[7] http://www.sciencedirect.com/science/article/pii/S0306261912001766

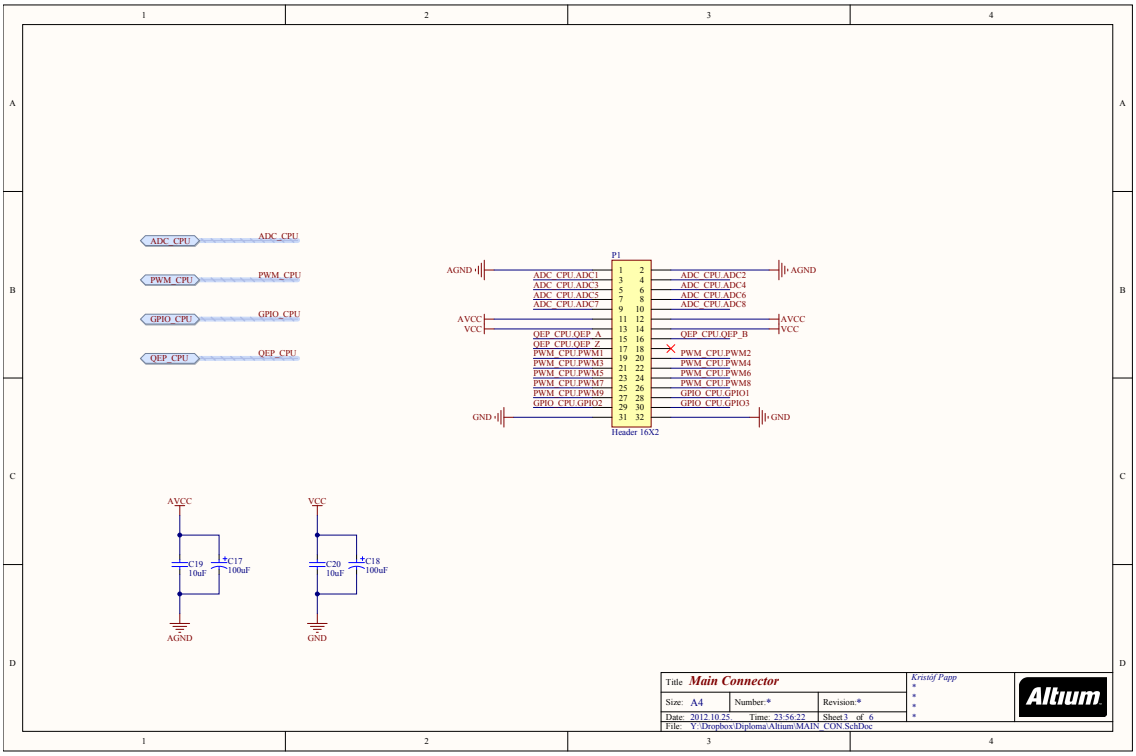
[8] http://www.ti.com/product/tms320f28035

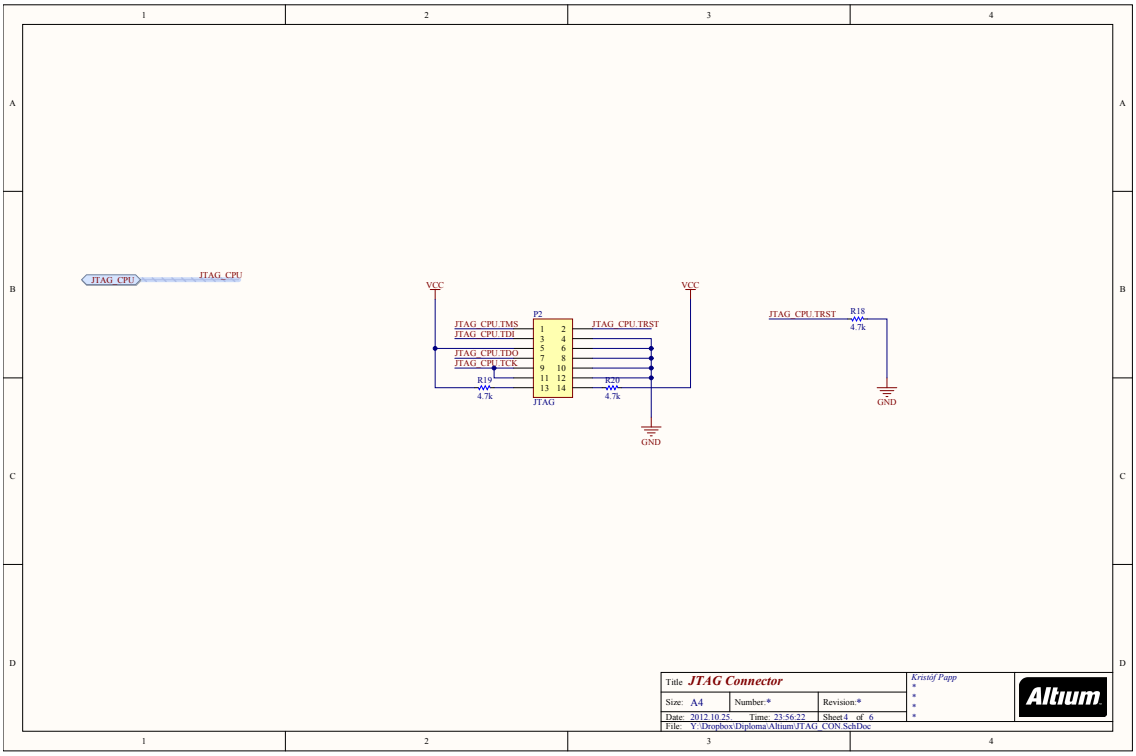
Függelék

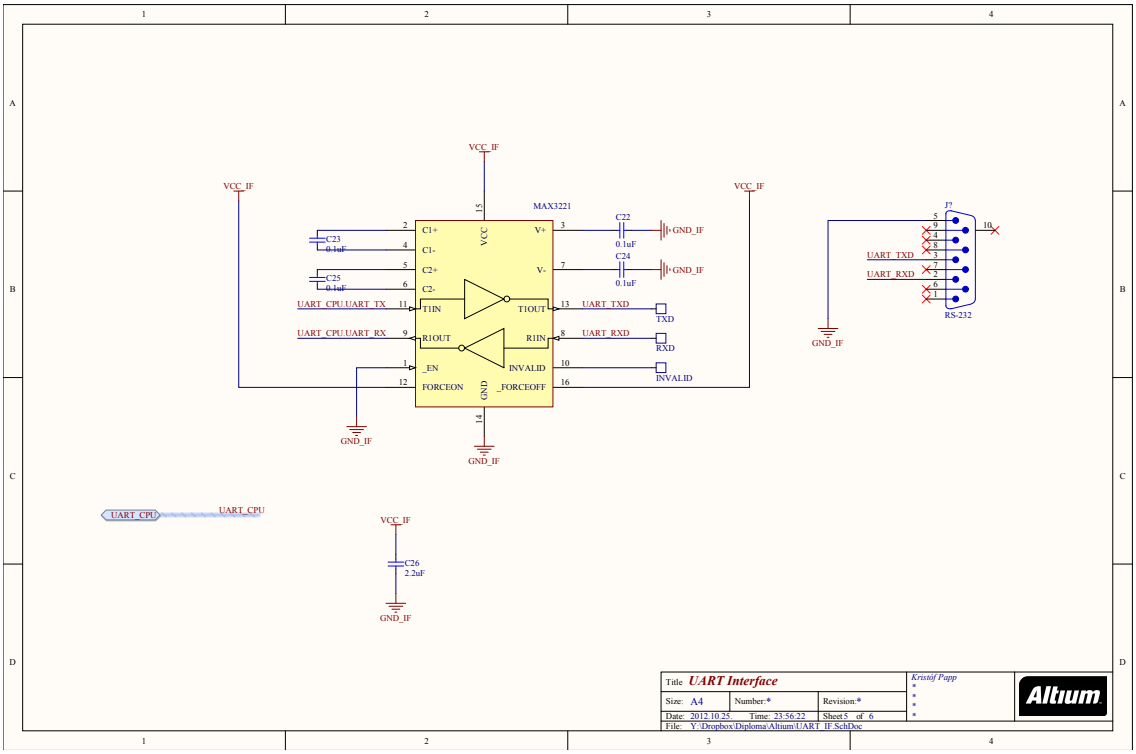
Az 500W-os prototípus kapcsolási rajza

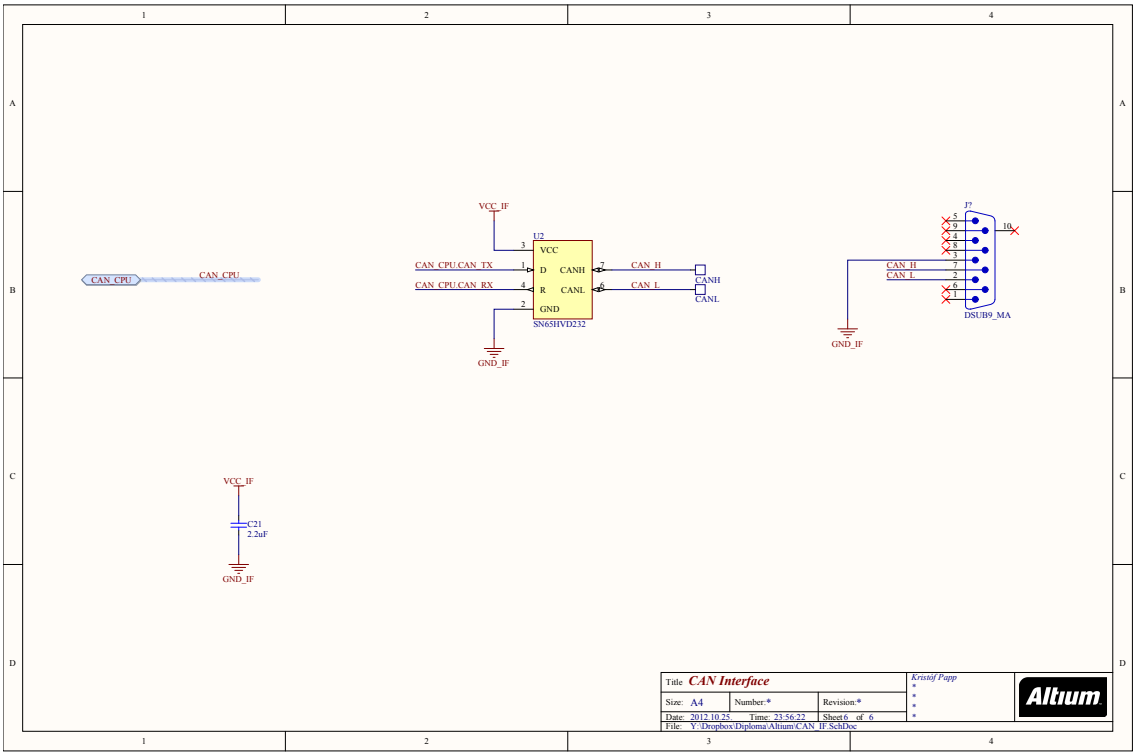


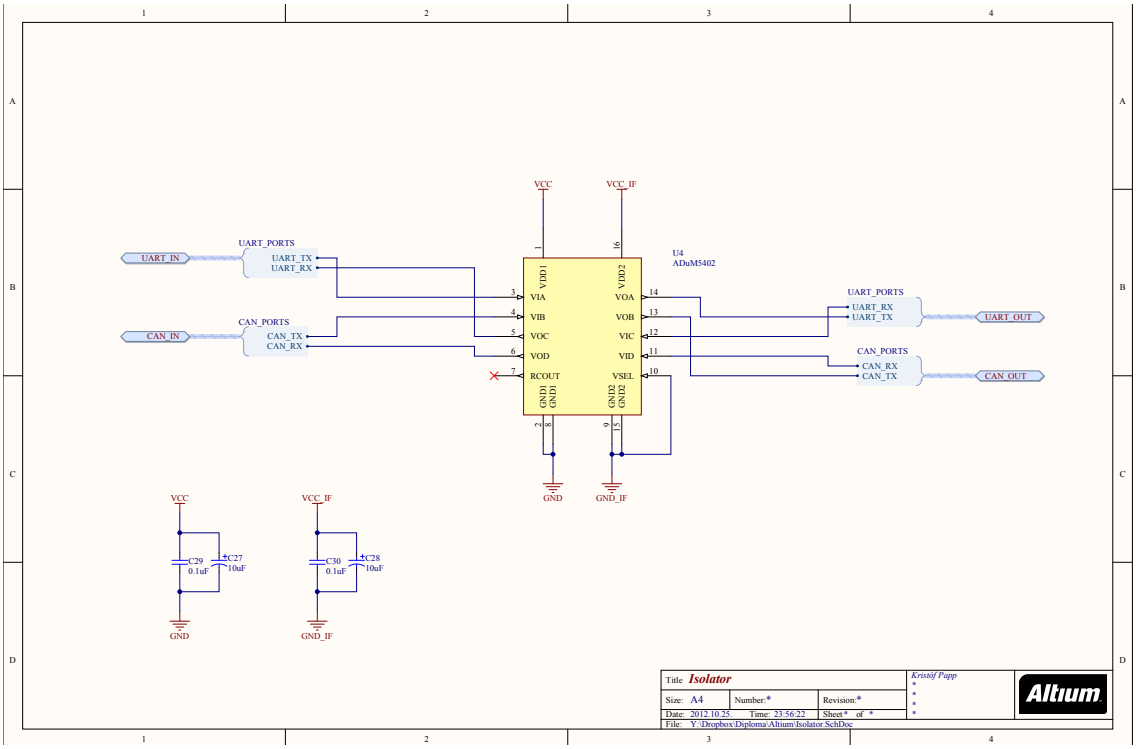


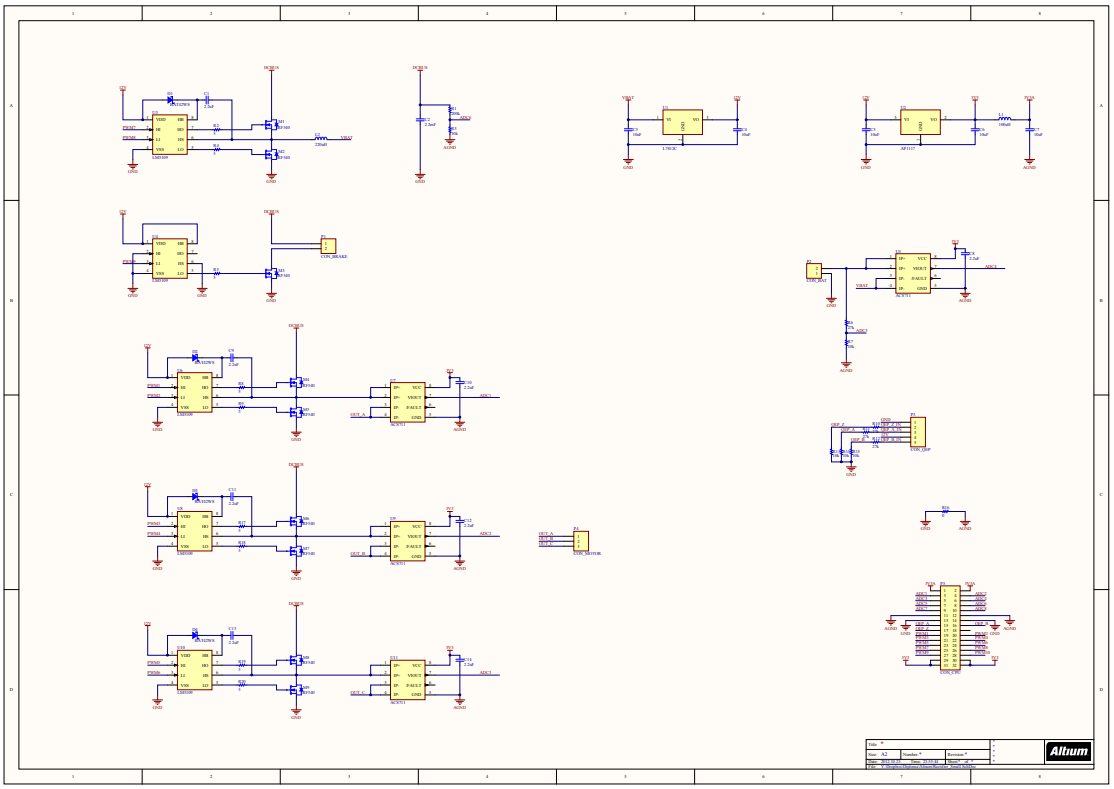

ADC_CPU
PWM_CPU
GPIO_CPU
QEP_CPU
P1
AGND
AVCC
VCC
GND
ADC_CPU.ADC1
ADC_CPU.ADC2
ADC_CPU.ADC3
ADC_CPU.ADC4
ADC_CPU.ADC5
ADC_CPU.ADC6
ADC_CPU.ADC7
ADC_CPU.ADC8
QEP_CPU.QEP_A
QEP_CPU.QEP_B
QEP_CPU.QEP_Z
PWM_CPU.PWM1
PWM_CPU.PWM2
PWM_CPU.PWM3
PWM_CPU.PWM4
PWM_CPU.PWM5
PWM_CPU.PWM6
PWM_CPU.PWM7
PWM_CPU.PWM8
PWM_CPU.PWM9
GPIO_CPU.GPIO1
GPIO_CPU.GPIO2
GPIO_CPU.GPIO3
Header 16X2
C19 10uF
C17 100uF
C20 10uF
C18 100uF
Title Main Connector
Size: A4
Number:*
Revision:*
Date: 2012.10.25
Time: 23:56:22
Sheet 3 of 6
File: Y:\Dropbox\Diploma\Altium\MAIN_CON.SchDoc
Kristóf Papp
Altium


JTAG_CPU
P2
JTAG_CPU.TMS
JTAG_CPU.TDI
JTAG_CPU.TDO
JTAG_CPU.TCK
JTAG_CPU.TRST
R19 4.7k
R20 4.7k
R18 4.7k
JTAG
Title JTAG Connector
Size: A4
Number:*
Revision:*
Date: 2012.10.25
Time: 23:56:22
Sheet 4 of 6
File: Y:\Dropbox\Diploma\Altium\JTAG_CON.SchDoc
Kristóf Papp
Altium


VCC_IF
MAX3221
C1+
C1-
C2+
C2-
VCC
V+
V-
T1IN
T1OUT
R1OUT
R1IN
_EN
INVALID
FORCEON
_FORCEOFF
GND
C22
C24
0.1uF
C23
C25
GND_IF
UART_CPU.UART_TX
UART_CPU.UART_RX
UART_TXD
UART_RXD
TXD
RXD
INVALID
J7
RS-232
UART_CPU
C26
2.2uF
Title UART Interface
Size: A4
Number:*
Revision:*
Date: 2012.10.25
Time: 23:56:22
Sheet 5 of 6
File: Y:\Dropbox\Diploma\Altium\UART_IF.SchDoc
Kristóf Papp
Altium


CAN_CPU
U2
VCC
D
R
GND
CANH
CANL
SN65HVD232
CAN_CPU.CAN_TX
CAN_CPU.CAN_RX
CAN_H
CAN_L
J7
DSUB9_MA
VCC_IF
GND_IF
C21
2.2uF
Title CAN Interface
Size: A4
Number:*
Revision:*
Date: 2012.10.25
Time: 23:56:22
Sheet 6 of 6
File: Y:\Dropbox\Diploma\Altium\CAN_IF.SchDoc
Kristóf Papp
Altium


VCC
VCC_IF
UART_IN
UART_PORTS
UART_TX
UART_RX
CAN_IN
CAN_PORTS
CAN_TX
CAN_RX
U4
ADuM5402
VDD1
VIA
VIB
VOC
VOD
RCOUT
GND1
GND1
VDD2
VOA
VOB
VIC
VID
VSEL
GND2
GND2
UART_PORTS
UART_RX
UART_TX
UART_OUT
CAN_PORTS
CAN_RX
CAN_TX
CAN_OUT
GND
GND_IF
C29
0.1uF
C27
10uF
C30
0.1uF
C28
10uF
Title Isolator
Size: A4
Number:*
Revision:*
Date: 2012.10.25
Time: 23:56:22
Sheet * of *
File: Y:\Dropbox\Diploma\Altium\Isolator.SchDoc
Kristóf Papp
Altium



Demo alkalmazás

Main.c

```
#include "Device.h"
#include "Init.h"
#include "DSP2803x_common/include/DSP2803x_Cla_defines.h"

#define   GLOBAL_Q       20
#include "IQmathLib.h"

#include "svgen_mf.h"

//////////////////////////////////////////////
// Global parameters                        //
//////////////////////////////////////////////
#pragma DATA_SECTION(ADCFilter_coefs, "CpuToCla1MsgRAM");
float32 ADCFilter_coefs[2] = {0.5, 0.5};

#pragma DATA_SECTION(DCBooster_enable, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(DCBooster_target, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(DCBooster_pgain, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(DCBooster_igain, "CpuToCla1MsgRAM");
Uint16 DCBooster_enable  = 0;
float32 DCBooster_target = 1225.0;  // 1225 = 20V
float32 DCBooster_pgain  = 2.0;
float32 DCBooster_igain  = 0.0005;

#pragma DATA_SECTION(Regeneration_enable, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(Regeneration_minvoltage, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(Regeneration_maxcurrent, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(Regeneration_pgain, "CpuToCla1MsgRAM");
Uint16 Regeneration_enable      = 0;
float32 Regeneration_minvoltage = 1531.0;  // 1531 = 25V
float32 Regeneration_maxcurrent = 0.0;
float32 Regeneration_pgain  = 1.5;

#pragma DATA_SECTION(Brake_enable, "CpuToCla1MsgRAM");
#pragma DATA_SECTION(Brake_voltage, "CpuToCla1MsgRAM");
Uint16 Brake_enable   = 0;
float32 Brake_voltage = 1838.0;  // 1838 = 30V


//////////////////////////////////////////////
// CLA variables                            //
//////////////////////////////////////////////
#pragma DATA_SECTION(CLA_state, "Cla1RAM");
Uint16 CLA_state;

#pragma DATA_SECTION(ADC0_filtered, "Cla1ToCpuMsgRAM");
#pragma DATA_SECTION(ADC1_filtered, "Cla1ToCpuMsgRAM");
#pragma DATA_SECTION(ADC2_filtered, "Cla1ToCpuMsgRAM");
#pragma DATA_SECTION(ADC3_filtered, "Cla1ToCpuMsgRAM");
#pragma DATA_SECTION(ADC4_filtered, "Cla1ToCpuMsgRAM");
#pragma DATA_SECTION(ADC5_filtered, "Cla1ToCpuMsgRAM");
float32 ADC0_filtered;
float32 ADC1_filtered;
float32 ADC2_filtered;
float32 ADC3_filtered;
```

```
float32 ADC4_filtered;
float32 ADC5_filtered;

#pragma DATA_SECTION(DCBooster_istate, "Cla1RAM");
float32 DCBooster_istate;

#pragma DATA_SECTION(Power_backflow, "Cla1RAM");
Uint16 Power_backflow;


//////////////////////////////////////////////
// CPU variables                            //
//////////////////////////////////////////////
SVGENMF svgen_mf1 = SVGENMF_DEFAULTS;


//////////////////////////////////////////////
// Function prototypes                      //
//////////////////////////////////////////////
void scia_xmit(int a);
void scia_msg(char *msg);
interrupt void CLA_control_isr(void);



void main(void)
{

    SystemInit();

    PIEInit();

    GPIOInit();

    SCIInit();

    PWMInit();

    ADCInit();

    CLAInit();


    EALLOW;
    PieVectTable.EPWM1_INT = &CLA_control_isr;
    EDIS;

    PieCtrlRegs.PIEIER3.bit.INTx1 = 1;               // Enable INT 3.1 in
the PIE (EPWM1_INT)
    IER |= M_INT3;                                      // Enable CPU
Interrupt 3
    EINT;                                           // Enable Global
interrupt INTM
    ERTM;                                           // Enable Global
realtime interrupt DBGM

    // Reset the variables
    Cla1ForceTask8andWait();
```

```
    for(;;)
    {
//        scia_xmit(((int)ADC0_filtered) & 0x00FF);
//        scia_xmit((((int)ADC0_filtered)>>8) & 0x00FF);
//        scia_xmit(((int)ADC1_filtered) & 0x00FF);
//        scia_xmit((((int)ADC1_filtered)>>8) & 0x00FF);
//        scia_xmit(((int)ADC2_filtered) & 0x00FF);
//        scia_xmit((((int)ADC2_filtered)>>8) & 0x00FF);
//        scia_xmit(((int)ADC3_filtered) & 0x00FF);
//        scia_xmit((((int)ADC3_filtered)>>8) & 0x00FF);
//        scia_xmit(((int)ADC4_filtered) & 0x00FF);
//        scia_xmit((((int)ADC4_filtered)>>8) & 0x00FF);
//        scia_xmit(((int)ADC5_filtered) & 0x00FF);
//        scia_xmit((((int)ADC5_filtered)>>8) & 0x00FF);


    }

}



////////////////////////////////////////////////
// 150kHz interrupts:                         //
////////////////////////////////////////////////
// Slice1 | Slice2 | Slice3 | Slice4 | Slice5   //
//                                            //
// ADC    | ADC    | ADC    | ADC    | ADC+Ctrl //
// CLA_control_isr                  | ---      //
////////////////////////////////////////////////

////////////////////////////////////////////////
// 30kHz interrupt                            //
////////////////////////////////////////////////
interrupt void CLA_control_isr(void)
{
    svgen_mf1.Gain = _IQ(0.5);   // Pass inputs to svgen_mf1
    svgen_mf1.Freq = _IQ(1.0); // Pass inputs to svgen_mf1
    svgen_mf1.FreqMax = _IQ(0.0002);

    SVGENMF_MACRO(svgen_mf1); // Call compute macro for svgen_mf1

//    Ta1 = svgen_mf1.Ta;       // Access the outputs of svgen_mf1
//    Tb1 = svgen_mf1.Tb;       // Access the outputs of svgen_mf1
//    Tc1 = svgen_mf1.Tc;       // Access the outputs of svgen_mf1
    scia_xmit(_IQtoIQ1(svgen_mf1.Ta) & 0x00FF);
    scia_xmit(_IQtoIQ1(svgen_mf1.Tb) & 0x00FF);
    scia_xmit(_IQtoIQ1(svgen_mf1.Tc) & 0x00FF);


    // Clear the CLA1_INT2 interrupt flag
    EPwm1Regs.ETCLR.bit.INT = 1;
    PieCtrlRegs.PIEACK.bit.ACK3 = 1;
}



// Transmit a character from the SCI
```

```
void scia_xmit(int a)
{
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
    SciaRegs.SCITXBUF=a;

}

void scia_msg(char * msg)
{
    int i;
    i = 0;
    while(msg[i] != '\0')
    {
        scia_xmit(msg[i]);
        i++;
    }
}
```

Init.c

```
#include "Init.h"
#include "CLAShared.h"
#include "DSP2803x_common/include/DSP2803x_EPwm_defines.h"


// These are defined by the linker file
extern Uint16 Cla1funcsLoadStart;
extern Uint16 Cla1funcsLoadEnd;
extern Uint16 Cla1funcsRunStart;

void MemCopy(Uint16 *SourceAddr, Uint16* SourceEndAddr, Uint16* DestAddr)
{
    while(SourceAddr < SourceEndAddr)
    {
       *DestAddr++ = *SourceAddr++;
    }
    return;
}

void SystemInit()
{
    EALLOW;

    // Disable Watchdog
    SysCtrlRegs.WDCR = 0x0068;

    // ADC Calibration
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1; // Enable ADC peripheral clock
    (*Device_cal)();
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0; // Return ADC clock to original
state

    // Select XTAL Oscillator
    SysCtrlRegs.CLKCTL.bit.XTALOSCOFF = 0;     // Turn on XTALOSC
    SysCtrlRegs.CLKCTL.bit.XCLKINOFF = 1;      // Turn off XCLKIN
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRC2SEL = 0;  // Switch to external clock
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL = 1;    // Switch from INTOSC1 to
INTOSC2/ext clk
```

```
    SysCtrlRegs.CLKCTL.bit.WDCLKSRCSEL = 1;     // Switch Watchdog Clk Src
to external clock
    SysCtrlRegs.CLKCTL.bit.INTOSC2OFF = 1;     // Turn off INTOSC2
    SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 1;     // Turn off INTOSC1

    // PLL Initialization
    SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
    SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1; // Before setting PLLCR turn off
missing clock detect logic
    SysCtrlRegs.PLLCR.bit.DIV = DSP28_PLLCR;
    while (SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1) {}
    SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
    if ((DSP28_DIVSEL == 1) || (DSP28_DIVSEL == 2))
    {
        SysCtrlRegs.PLLSTS.bit.DIVSEL = DSP28_DIVSEL;
    }
    else if (DSP28_DIVSEL == 3)
    {
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
        DELAY_US(50L);
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 3;
    }

    // Low-Speed Peripherial Clock Setup
    SysCtrlRegs.LOSPCP.all = DSP28_LOSPCP;

    // Clock selection
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;      // ADC
    //SysCtrlRegs.PCLKCR3.bit.COMP1ENCLK = 1;    // COMP1
    //SysCtrlRegs.PCLKCR3.bit.COMP2ENCLK = 1;    // COMP2
    //SysCtrlRegs.PCLKCR3.bit.COMP3ENCLK = 1;    // COMP3
    //SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1;    // eCAP1
    SysCtrlRegs.PCLKCR0.bit.ECANAENCLK = 1;    // eCAN-A
    SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 1;    // eQEP1
    SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;    // ePWM1
    SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;    // ePWM2
    SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;    // ePWM3
    SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;    // ePWM4
    SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;    // ePWM5
    SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;    // ePWM6
    //SysCtrlRegs.PCLKCR1.bit.EPWM7ENCLK = 1;    // ePWM7
    //SysCtrlRegs.PCLKCR0.bit.HRPWMENCLK = 1;    // HRPWM
    //SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;     // I2C
    //SysCtrlRegs.PCLKCR0.bit.LINAENCLK = 1;     // LIN-A
    SysCtrlRegs.PCLKCR3.bit.CLA1ENCLK = 1;     // CLA1
    SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 1;     // SCI-A
    //SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 1;     // SPI-A
    //SysCtrlRegs.PCLKCR0.bit.SPIBENCLK = 1;     // SPI-B

    EDIS;
}

void PIEInit()
{
    EALLOW;

    // Disable Interrupts at the CPU level:
    DINT;
```

```
    // Disable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;

    // Clear all PIEIER registers:
    PieCtrlRegs.PIEIER1.all = 0;
    PieCtrlRegs.PIEIER2.all = 0;
    PieCtrlRegs.PIEIER3.all = 0;
    PieCtrlRegs.PIEIER4.all = 0;
    PieCtrlRegs.PIEIER5.all = 0;
    PieCtrlRegs.PIEIER6.all = 0;
    PieCtrlRegs.PIEIER7.all = 0;
    PieCtrlRegs.PIEIER8.all = 0;
    PieCtrlRegs.PIEIER9.all = 0;
    PieCtrlRegs.PIEIER10.all = 0;
    PieCtrlRegs.PIEIER11.all = 0;
    PieCtrlRegs.PIEIER12.all = 0;

    // Clear all PIEIFR registers:
    PieCtrlRegs.PIEIFR1.all = 0;
    PieCtrlRegs.PIEIFR2.all = 0;
    PieCtrlRegs.PIEIFR3.all = 0;
    PieCtrlRegs.PIEIFR4.all = 0;
    PieCtrlRegs.PIEIFR5.all = 0;
    PieCtrlRegs.PIEIFR6.all = 0;
    PieCtrlRegs.PIEIFR7.all = 0;
    PieCtrlRegs.PIEIFR8.all = 0;
    PieCtrlRegs.PIEIFR9.all = 0;
    PieCtrlRegs.PIEIFR10.all = 0;
    PieCtrlRegs.PIEIFR11.all = 0;
    PieCtrlRegs.PIEIFR12.all = 0;

    // Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

    // Enable the PIE Vector Table
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;

    EDIS;
}

void GPIOInit()
{
    EALLOW;

    GpioCtrlRegs.GPAPUD.all = 0xFFFFFFFF;
    GpioCtrlRegs.GPBPUD.all = 0xFFFFFFFF;

    // LEDs
    GpioDataRegs.GPADAT.bit.GPIO17 = 0;  // GPIO17: low
    GpioDataRegs.GPADAT.bit.GPIO18 = 0;  // GPIO18: low
    GpioCtrlRegs.GPADIR.bit.GPIO17 = 1;  // GPIO17: output
    GpioCtrlRegs.GPADIR.bit.GPIO18 = 1;  // GPIO18: output

    EDIS;
}

void SCIInit()
{
```

```
    EALLOW;

    // Pin Init
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 1;    // Enable transmitter IC
    GpioDataRegs.GPADAT.bit.GPIO27 = 0;    // Enable transmitter IC
    GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;     // Enable pull-up for GPIO28
(SCIRXDA)
    GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;     // Enable pull-up for GPIO29
(SCITXDA)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO28  =  3;     //  Asynch  input  GPIO28
(SCIRXDA)
    GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;   // Configure GPIO28 for SCIRXDA
operation
    GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;   // Configure GPIO29 for SCITXDA
operation

    // SCI FIFO Init
    SciaRegs.SCIFFTX.all = 0xE040;
    SciaRegs.SCIFFRX.all = 0x2044;
    SciaRegs.SCIFFCT.all = 0x0;

    // SCI Init
    SciaRegs.SCICCR.bit.ADDRIDLE_MODE = 0;  // IDLE-Line mode
    SciaRegs.SCICCR.bit.LOOPBKENA = 0;      // Disable Loopback mode
    SciaRegs.SCICCR.bit.PARITYENA = 0;      // No Parity bit
    SciaRegs.SCICCR.bit.SCICHAR = 7;        // 8 bits
    SciaRegs.SCICCR.bit.STOPBITS = 0;       // 1 STOP bit
    SciaRegs.SCICTL1.bit.SWRESET = 0;       // Reset state-machine
    SciaRegs.SCICTL1.bit.TXENA = 1;         // Enable TX
    SciaRegs.SCICTL1.bit.RXENA = 1;         // Enable RX
    SciaRegs.SCICTL2.bit.TXINTENA = 0;      // Disable TX Interrupt
    SciaRegs.SCICTL2.bit.RXBKINTENA = 0;    // Disable RX Interrupt
    SciaRegs.SCIHBAUD = 0;                  // 57600 baud @LSPCLK = 30MHz
    SciaRegs.SCILBAUD = 64;
    SciaRegs.SCICTL1.bit.SWRESET = 1;       // Restart SCI from reset

    EDIS;
}

void PWMInit()
{
    EALLOW;

    // Init PWM outputs as LOW for safety
    GpioCtrlRegs.GPADIR.bit.GPIO0 = 1;   // PWM1
    GpioCtrlRegs.GPADIR.bit.GPIO1 = 1;   // PWM2
    GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;   // PWM3
    GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;   // PWM4
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;  // PWM5
    GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;  // PWM6
    GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;   // PWM7
    GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;   // PWM8
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;   // PWM9

    GpioDataRegs.GPADAT.bit.GPIO0 = 0;   // PWM1 - CH1 - high (EPWM1A)
    GpioDataRegs.GPADAT.bit.GPIO1 = 1;   // PWM2 - CH1 - low (EPWM1B)
    GpioDataRegs.GPADAT.bit.GPIO4 = 0;   // PWM3 - CH2 - high (EPWM3A)
    GpioDataRegs.GPADAT.bit.GPIO5 = 1;   // PWM4 - CH2 - low (EPWM3B)
    GpioDataRegs.GPADAT.bit.GPIO11 = 0;  // PWM5 - CH3 - high (EPWM6B)
```

```
    GpioDataRegs.GPADAT.bit.GPIO10 = 1;  // PWM6 - CH3 - low (EPWM6A)
    GpioDataRegs.GPADAT.bit.GPIO2 = 0;   // PWM7 - buck (EPWM2A)
    GpioDataRegs.GPADAT.bit.GPIO3 = 0;   // PWM8 - boost (EPWM2B)
    GpioDataRegs.GPADAT.bit.GPIO6 = 0;   // PWM9 - brake (EPWM4A)

    // Pin Init
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // Configure GPIO0 as EPWM1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;   // Configure GPIO1 as EPWM1B
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;   // Configure GPIO4 as EPWM3A
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;   // Configure GPIO5 as EPWM3B
    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;  // Configure GPIO11 as EPWM6B
    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;  // Configure GPIO10 as EPWM6A
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // Configure GPIO2 as EPWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // Configure GPIO3 as EPWM2B
//    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;   // Configure GPIO6 as EPWM4A


    ////////////////////////////////////////////////
    // Setup EPWM1 (CH1 - 30kHz)                  //
    ////////////////////////////////////////////////
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;           // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm1Regs.TBPRD = 1000;                          // Set timer period
    EPwm1Regs.CMPA.half.CMPA = 1000;                    // Set Compare A
value
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
    EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm1Regs.TBCTR = 0x0000;                        // Clear counter

    EPwm1Regs.DBCTL.bit.POLSEL  =  DB_ACTV_HIC;            //  Active  high
complementary
    EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    EPwm1Regs.DBRED = 6;                              // Rising edge delay
(0.1us)
    EPwm1Regs.DBFED = 6;                             // Falling edge delay
(0.1us)

    EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;   // Select ADCSOCA on Zero
event
    EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;         // Generate ADCSOCA on 1st
event
    EPwm1Regs.ETSEL.bit.INTEN = 1;             // Enable ADCSOCA


    ////////////////////////////////////////////////
    // Setup EPWM2 (DC/DC - 30kHz)                //
    ////////////////////////////////////////////////
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;           // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm2Regs.TBPRD = 1000;                          // Set timer period
    EPwm2Regs.CMPA.half.CMPA = 1000;                 // Set Compare A value
    EPwm2Regs.CMPB = 1000;                           // Set Compare B value
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm2Regs.AQCTLB.bit.CBU = AQ_SET;
```

```
    EPwm2Regs.AQCTLB.bit.CBD = AQ_CLEAR;
    EPwm2Regs.TBCTR = 0x0000;                       // Clear counter


    //////////////////////////////////////////////
    // Setup EPWM3 (CH2 - 30kHz)                  //
    //////////////////////////////////////////////
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;          // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm3Regs.TBPRD = 1000;                         // Set timer period
    EPwm3Regs.CMPA.half.CMPA = 1000;                // Set Compare A value
    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
    EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm3Regs.TBCTR = 0x0000;                       // Clear counter

    EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;           // Active high
complementary
    EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    EPwm3Regs.DBRED = 6;                              // Rising edge delay
(0.1us)
    EPwm3Regs.DBFED = 6;                             // Falling edge delay
(0.1us)


    //////////////////////////////////////////////
    // Setup EPWM4 (Brake - 30kHz)                //
    //////////////////////////////////////////////
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;          // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm4Regs.TBPRD = 1000;                         // Set timer period
    EPwm4Regs.CMPA.half.CMPA = 1000;                // Set Compare A value
    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
    EPwm4Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm4Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm4Regs.TBCTR = 0x0000;                       // Clear counter


    //////////////////////////////////////////////
    // Setup EPWM5 (ADC timer - 150kHz)           //
    //////////////////////////////////////////////
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm5Regs.TBPRD = 399;                       // Set timer period
    EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;  // Count up
    EPwm5Regs.TBCTR = 0x0000;                    // Clear counter

    EPwm5Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;  // Select ADCSOCA on Zero
event
    EPwm5Regs.ETPS.bit.SOCAPRD = ET_1ST;        // Generate ADCSOCA on 1st
event
    EPwm5Regs.ETSEL.bit.SOCAEN = 1;             // Enable ADCSOCA


    //////////////////////////////////////////////
    // Setup EPWM6 (CH3 - 30kHz)                  //
```

```
    ////////////////////////////////////////////
    EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;           // Clock ratio to
SYSCLKOUT: TBCLK = SYSCLKOUT / (HSPCLKDIV × CLKDIV)
    EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm6Regs.TBPRD = 1000;                          // Set timer period
    EPwm6Regs.CMPA.half.CMPA = 1000;                 // Set Compare A value
    EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Symmetrical mode
    EPwm6Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm6Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    EPwm6Regs.TBCTR = 0x0000;                        // Clear counter

    EPwm6Regs.DBCTL.bit.POLSEL  =  DB_ACTV_LOC;             //  Active  low
complementary
    EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    EPwm6Regs.DBRED = 6;                               // Rising edge delay
(0.1us)
    EPwm6Regs.DBFED = 6;                              // Falling edge delay
(0.1us)


    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;      // Start all counters


    EDIS;
}

void ADCInit()
{
    EALLOW;

    // ADC
    AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;      // Power ADC BG
    AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;      // Power reference
    AdcRegs.ADCCTL1.bit.ADCPWDN   = 1;      // Power ADC
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;      // Enable ADC
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;      // Select interal BG
    DELAY_US(1000L);

    AdcRegs.ADCSOC0CTL.bit.CHSEL   = 0;      // Set SOC0 channel select to
ADCINA0 - Current1
    AdcRegs.ADCSOC1CTL.bit.CHSEL   = 1;      // Set SOC1 channel select to
ADCINA1 - Current2
    AdcRegs.ADCSOC2CTL.bit.CHSEL   = 2;      // Set SOC2 channel select to
ADCINA2 - Current3
    AdcRegs.ADCSOC3CTL.bit.CHSEL   = 3;      // Set SOC3 channel select to
ADCINA3 - CurrentBat
    AdcRegs.ADCSOC4CTL.bit.CHSEL   = 4;      // Set SOC4 channel select to
ADCINA4 - VBat
    AdcRegs.ADCSOC5CTL.bit.CHSEL   = 5;      // Set SOC5 channel select to
ADCINA5 - VDCBus
    AdcRegs.ADCSOC0CTL.bit.ACQPS  = 6;      // Set SOC0 conversion time to
20 ADCCLK
    AdcRegs.ADCSOC1CTL.bit.ACQPS  = 6;      // Set SOC1 conversion time to
20 ADCCLK
    AdcRegs.ADCSOC2CTL.bit.ACQPS  = 6;      // Set SOC2 conversion time to
20 ADCCLK
    AdcRegs.ADCSOC3CTL.bit.ACQPS  = 6;      // Set SOC3 conversion time to
20 ADCCLK
```

```
    AdcRegs.ADCSOC4CTL.bit.ACQPS  = 6;      // Set SOC4 conversion time to
20 ADCCLK
    AdcRegs.ADCSOC5CTL.bit.ACQPS  = 6;      // Set SOC5 conversion time to
20 ADCCLK
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC0
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC1
    AdcRegs.ADCSOC2CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC2
    AdcRegs.ADCSOC3CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC3
    AdcRegs.ADCSOC4CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC4
    AdcRegs.ADCSOC5CTL.bit.TRIGSEL = 0x0D; // ePWM5 will trigger SOC5
    AdcRegs.INTSEL1N2.bit.INT1SEL = 5;     // Connect ADCINT1 to EOC5
    AdcRegs.INTSEL1N2.bit.INT1CONT = 1;     // ADCINT1 does not need to be
cleared
    AdcRegs.INTSEL1N2.bit.INT1E = 1;       // Enable ADCINT1
    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;   // Late interrupt generation

    EDIS;
}

void CLAInit()
{
    EALLOW;

    MemCopy(&Cla1funcsLoadStart, &Cla1funcsLoadEnd, &Cla1funcsRunStart);

    Cla1Regs.MVECT1         =         (Uint16)        (&Cla1Task1        -
&Cla1Prog_Start)*sizeof(Uint32);
    Cla1Regs.MVECT2         =         (Uint16)        (&Cla1Task2        -
&Cla1Prog_Start)*sizeof(Uint32);
    Cla1Regs.MVECT8         =         (Uint16)        (&Cla1Task8        -
&Cla1Prog_Start)*sizeof(Uint32);
    Cla1Regs.MMEMCFG.bit.PROGE = 1;           // Map CLA program memory to
the CLA
    Cla1Regs.MMEMCFG.bit.RAM1E = 1;          // Map CLA data memory to the
CLA
    Cla1Regs.MPISRCSEL1.bit.PERINT1SEL = 0;  // ADCINT1 triggers Task1
    Cla1Regs.MPISRCSEL1.bit.PERINT2SEL = 1;  // Software source
    Cla1Regs.MPISRCSEL1.bit.PERINT8SEL = 1;  // Software source
    Cla1Regs.MCTL.bit.IACKE = 1;             // Enable IACK to start tasks
via software
    Cla1Regs.MIER.bit.INT1 = 1;              // Enable Task 1
    Cla1Regs.MIER.bit.INT2 = 1;              // Enable Task 2
    Cla1Regs.MIER.bit.INT8 = 1;              // Enable Task 8

    EDIS;
}
```

CLAShared.h

```
#ifndef CLA_SHARED_H
#define CLA_SHARED_H

#ifdef __cplusplus
extern "C" {
#endif

    #include "Device.h"

    //////////////////////////////////////////////
    // Global parameters                        //
```

```
    ////////////////////////////////////////////////
    extern float32 ADCFilter_coefs[2];

    extern Uint16 DCBooster_enable;
    extern float32 DCBooster_target;
    extern float32 DCBooster_pgain;
    extern float32 DCBooster_igain;

    extern Uint16 Regeneration_enable;
    extern float32 Regeneration_minvoltage;
    extern float32 Regeneration_maxcurrent;
    extern float32 Regeneration_pgain;

    extern Uint16 Brake_enable;
    extern float32 Brake_voltage;
    ////////////////////////////////////////////////
    // CLA variables                                //
    ////////////////////////////////////////////////
    extern Uint16 CLA_state;

    extern float32 ADC0_filtered;
    extern float32 ADC1_filtered;
    extern float32 ADC2_filtered;
    extern float32 ADC3_filtered;
    extern float32 ADC4_filtered;
    extern float32 ADC5_filtered;

    extern float32 DCBooster_istate;

    extern Uint16 Power_backflow;

    // The following are symbols defined in the CLA assembly code
    // Including them in the shared header file makes them
    // .global and the main CPU can make use of them.

    extern Uint32 Cla1Prog_Start;
    extern Uint32 Cla1Task1;
    extern Uint32 Cla1Task2;
    extern Uint32 Cla1Task3;
    extern Uint32 Cla1Task4;
    extern Uint32 Cla1Task5;
    extern Uint32 Cla1Task6;
    extern Uint32 Cla1Task7;
    extern Uint32 Cla1Task8;
    extern Uint32 Cla1T1End;
    extern Uint32 Cla1T2End;
    extern Uint32 Cla1T3End;
    extern Uint32 Cla1T4End;
    extern Uint32 Cla1T5End;
    extern Uint32 Cla1T6End;
    extern Uint32 Cla1T7End;
    extern Uint32 Cla1T8End;

#ifdef __cplusplus
}
#endif /* extern "C" */

#endif  // end of CLA_SHARED definition
```