Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

Hermann Izsák

Akkumulátor felügyeleti rendszer mikroprocesszorának cseréje a funkcionalitás megőrzése mellett

KONZULENS

Dr. Renczes Balázs

BUDAPEST, 2020

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2020. 05. 21.

...…….

Hermann Izsák

SZAKDOLGOZAT FELADAT

Hermann Izsák
Villamosmérnök hallgató részére

Akkumulátor felügyeleti rendszer mikroprocesszorának cseréje a funkcionalitás megőrzése mellett

A lítium-ion kémiájú akkumulátorok egyik hátránya, hogy érzékenyek a töltés és kisütés paramétereire, különös tekintettel a túltöltésre, a mélykisütésre, a túl alacsony, illetve túl magas hőmérsékletekre. Ezek ellen nyújt védelmet az akkumulátor felügyeleti rendszer (BMS), ami folyamatosan figyelve a cellák töltöttségét és hőmérsékletét, szükség esetén megfelelően beavatkozik az akkumulátor működésébe, ezzel biztonságos használatot tesz lehetővé, továbbá növelheti az akkumulátor élettartamát

Mivel a megbízó cég (Xtalin Kft.) már minden eszközében ARM architektúrájú, STM családba tartozó mikroprocesszort használ, ezért a korábban kifejlesztett BMS modulokban megtalálható AVR alapú Atmel mikroprocesszort lecserélik. Így az STM mikrovezérlőcsaládra egységesen fejlesztett szoftvercsomagokat minden eszközön ugyanúgy fel lehet majd használni, továbbá a szoftverek fejlesztése és karbantartása is egyszerűsödik a processzorcsaládok számának csökkentésével.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a feladat végrehajtásához szükséges elvégzendő részfeladatokat!
- Elemezze a mikroprocesszor cseréjével járó, körülötte lévő áramkörökben eszközölendő változtatásokat, hiánypótlásokat!
- Készítsen megfelelően működő szoftvert az új mikroprocesszoros rendszerhez!
- Igazolja a BMS funkcióinak működését, hasonlítsa össze az esetleges eltéréseket!

Tanszéki konzulens: Dr. Renczes Balázs, adjunktus
Külső konzulens: Szabó Ádám (Xtalin Kft.)

Budapest, 2020.05.21.

……..
Dr. Dabóczi Tamás
tanszékvezető
egyetemi tanár, DSc

Összefoglaló

Ezen szakdolgozat keretében egy – az Xtalin Kft által tervezett és forgalmazott – akkumulátor-felügyeleti rendszer mikroprocesszorának cseréje miatt szükséges áramköri átalakítások, szoftveres újratervezés és megvalósítás kerül bemutatásra. A projekt motivációjára már a bevezető fejezetben is kitérek, ahol a szakterületet kevésbé ismerők számára fejtem ki a téma fontosabb részeit. A későbbi fejezetekben részletekbe menőbben tárgyalom a problémák egyes elemeit, úgy, mint a régi és az új mikroprocesszor felépítésének és az egyes perifériáinak különbségeit, az ezekből adódó szoftveres, illetve hardveres módosítások szükségességét és az ezekkel járó nehézségeket, kompromisszumokat vagy éppen fejlesztési lehetőségeket. Külön fejezetben foglalkozom a kész projekt tesztelésével, illetve a tesztelés során felmerülő hibákkal és ezen hibák elhárításával. A tesztelésről szóló fejezetet követően kitérek a fejlesztési lehetőségekre is, melynek során nagyvonalakban ismertetem az esetleges későbbi fejlesztések megvalósítási módját és motivációját.

Abstract

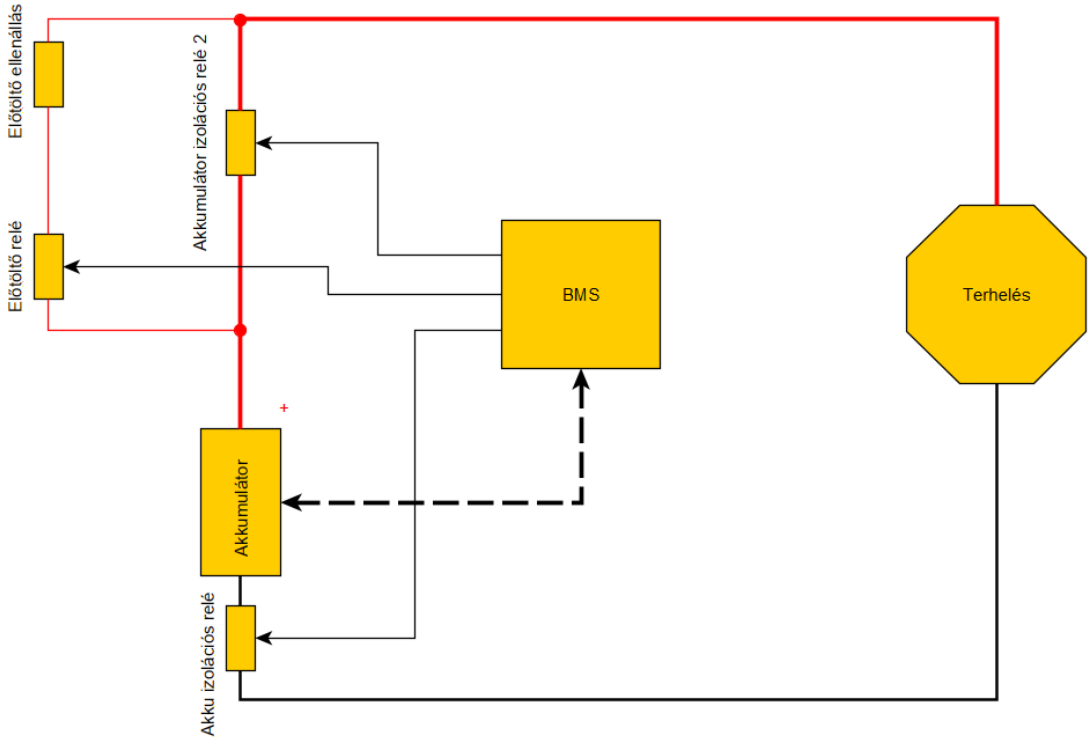
This thesis presents the circuit modifications, software redesign, and implementation required to replace the microprocessor of a BMS (Battery Management System) designed and promoted by Xtalin Ltd. I cover the motivation of this project already in the Introduction, where I explain the topic's more important parts to the less informed readers. In the later chapters, I also discuss some of the problems' parts' more in detail, such as the differences between the old and the new microprocessors' peripherals and architectures, the resulting hardware and software-modifications needed, and the related difficulties, compromises or even development opportunities. I have written a separate chapter about the completed project's testing, in which I also mention the issues that emerged during it following which I tell the process of troubleshooting in detail. Following the chapter about testing, I also write about the improvement possibilities, during which I broadly describe their method of the realizations as well as the motivations.

1 Bevezetés

Napjainkban rohamosan terjednek a különböző elektromos járművek, illetve hordozható műszerek, melyek általában hosszú akkumulátor-üzemidőt követelnek meg. Ezt a követelményt a lítium-ion kémiájú akkumulátorok kiválóan tudják teljesíteni, így ezekben az alkalmazásokban legtöbbször ezt az akkumulátor típust használjuk. Azonban a nagy kapacitás mellett kompromisszumokat kell kötnünk, mivel a lítium-ionos akkumulátorok nagyon érzékenyek a töltés, illetve a kisütés paramétereire. Ez annyit jelent, hogy nem tölthetjük és nem is süthetjük ki akármekkora árammal, továbbá nem tölthetjük és nem meríthetjük egy bizonyos feszültségszint és hőmérsékletszint felé, illetve alá. Ezeket a szabályokat rendszeresen vagy akár egyszeri alkalommal is megszegve súlyosan is károsíthatjuk a cellákat, ami már nem csak a cella élettartamát veszélyeztetheti (kapacitás csökkenése), hanem a cella környezetében lévő műszerek és személyek testi épségét is (robbanásveszély, tűzveszély). Ezekből következik, hogy az ilyen akkumulátor típusoknak a töltési, illetve kisütési paramétereit szabályozni kell. Ezt a feladatot látja el az akkumulátor felügyeleti rendszer (továbbiakban csak BMS mint Battery Management System).

Ahhoz, hogy a BMS ezt a feladatot el tudja végezni, monitoroznia kell az egyes cellák feszültségét és hőmérsékletét, melyek alapján szükség esetén be kell avatkoznia a töltésbe, illetve a kisütésbe. A legnagyobb megengedett feszültség elérésekor meg kell akadályoznia a töltést, a legalacsonyabb megengedett feszültség esetén pedig a merítést. Ezen felül a cellák töltöttsége közötti különbséget is ki kell egyenlítenie, ami pedig az akkumulátorcsomag maximális töltöttségét szolgálja. Ennek a feladatnak a szükségessége az egyes cellák nem tökéletes egyformaságából adódik, melynek köszönhetően, még ha ugyanolyan típusú is két cella, előfordulhat, hogy más ütemben töltődnek. A különböző ütemben töltődő cellák esetén pedig előfordulhat az, hogy egy cella már teljesen feltöltődött, a többi pedig még nem, melynek esetében nyilván nem állhat le a töltés. A cellák hőmérsékletét figyelve pedig tudnunk kell szabályozni az akkumulátor csomag hűtését és terhelését. Ha túlmelegedik, akkor csökkenteni kell a belőle kivett teljesítményt és/vagy növelni a hűtést. A BMS feladata az is, hogy valamilyen formában tájékoztassa a rendszer többi részét, illetve a felhasználót az akkumulátor állapotáról. E miatt fontos, hogy a státusz adatokat kommunikálni tudja a BMS valamilyen protokoll szerint. Az autóiparban elterjedt a CAN busz használata erre a célra. A CAN buszon keresztül tudjuk tájékoztatni a rendszer többi részét, illetve a felhasználót, többek között az akkumulátor

csomag töltöttségi szintjéről, biztonsági állapotáról és a még kivehető teljesítményről. Szintén a BMS felelős az elektromos autóknál és járműveknél elterjedt regeneratív fékezés vezérléséért. Továbbá előfordulhat, hogy az elektromos járműben földelési hiba áll elő, ilyenkor a BMS-nek ennek megfelelően kell beavatkoznia, általában le kell kapcsolnia a fő akkumulátort. Ebből következik, hogy a BMS-nek folyamatosan monitoroznia kell a földelési ellenállást, de legalábbis mindenképpen reagálnia kell a hibára. Az akkumulátor csatlakoztatása a terheléshez csak azután történhet, hogy a terhelés belső kapacitásai (ez egy elektromos motorvezérlőnél jelentős lehet) feltöltődtek egy előtöltő áramkörön keresztül, így megakadályozva a hirtelen nagy bemeneti áramokat, ami káros lehet a műszerekre. Ezt a problémát az úgynevezett precharge-áramkörrel (1.1 ábra) oldjuk meg, melynek vezérléséért szintén a BMS felelős.



1.1. ábra: A precharge áramkör és környezete

A fentiek alapján kimondhatjuk, hogy a BMS áramköröknek, számos fontos feladatot kell ellátniuk, mely az akkumulátor élettartamának megőrzése szempontjából elengedhetetlen is lehet.

Ezen szakdolgozat keretein belül, egy az Xtalin Kft. által biztosított BMS modul továbbfejlesztését tárgyalom, ami a mikroprocesszor kicserélését és az ezzel járó hardveres és szoftveres módosításokat takarja. Fontos tudni, hogy az eredeti BMS modul

tökéletesen működőképes mindenféle hardveres, illetve szoftveres módosítások híján is, ezért lényegi kérdés a mikroprocesszor cseréjével kapcsolatos döntés oka. A válasz egyszerűen összefoglalható: továbbfejlesztési lehetőség és szoftveres kompatibilitás. A továbbfejlesztési lehetőséget szolgálja az a tény, hogy az új mikroprocesszor a régivel ellentétben nagyobb számítási kapacitással bír és mindemellett több, nagyobb sebességű perifériái állnak rendelkezésre, így például egy esetleges jövőbeni szoftveres funkcióbővítés nem fog a sebesség miatt akadályba ütközni. Továbbá egy esetleges hardveres bővítésnél, nagyobb valószínűséggel lesz kisebb módosítás is elegendő a NYÁK-kon. A szoftveres kompatibilitás pedig abból adódik, hogy a megbízó cég többi eszköze is az új mikroprocesszorral megegyező típusú processzorral működik, így egy módisítás vagy bővítés esetén más eszközök szoftveréből felhasználhatók egyes részek, ezzel pedig a szoftver karbantarthatósága is nő. Ezt a problémát úgy is beláthatjuk, hogy belegondolunk abba, hogy a processzorcsere nélkülözése esetén két alapvetően különböző mikroprocesszor-típusra kell szoftvert fejleszteni, ami az alsóbb szoftverrétegek megírásánál többletmunkát jelent. A többletmunka abból adódik, hogy a két különböző típusú processzor ugyanazon perifériájának használata, de akár az inicializációja is teljesen más lehet az architekturális különbségekből adódóan. Ez a különbség pedig azt jelenti, hogy például a két különböző típusú processzor egyes regiszterei teljesen más memóriaterületen foglalnak helyet, így máshogy kell őket címezni. További példa az architekturális különbségre az, ha az egyik processzor perifériája kevésbé automatizált, mint a másiké. Előfordulhat ugyanis, hogy például egy SPI üzenet kiküldésére – az egyik processzor esetében – elegendő az üzenetet egy adott regiszterbe beírni, ami ezt követően automatikusan kiküldésre kerül, illetve ugyanezen periféria egy másik processzor esetében működhet úgy is, hogy az üzenet az adott regiszterbe való beírását követően, valamilyen vezérlőregiszterbe (az adatlap alapján) egy bizonyos bitkombinációt be kell írni, ahhoz, hogy az üzenet küldése megindduljon. Ezt a későbbi fejezetekben is láthatjuk majd.

2 Az Xtalin Kft. által biztosított BMS bemutatása

A következőkben bemutatásra kerül a feladatban említett BMS modul, melyet az Xtalin Kft. biztosított.

A szakdolgozat során használt BMS modul 18 lítiumos cella feszültségének legalább 10mV felbontású mérésére, és legalább 4 külső hőmérővel hőmérséklet mérésre 1°C-os felbontással képes, valamint CAN 2.0A és CAN 2.0B kompatibilis interfésszel rendelkezik, visszaméri a modul tápfeszültségét, illetve méri a PCB hőmérsékletét 1°C-os felbontással. Fontos tudni, hogy a cég által biztosított BMS modell moduláris, vagyis lehetőség van több BMS modul (maximum 8) összekapcsolására. Ekkor viszont a modulok kis- és "nagyfeszültségű" oldalai között akár több száz voltos feszültség is megjelenhet, ezért közöttük található egy izoláció, amely maximum 600V-ig nyújt védelmet.

A cellafeszültségek mérésére – a cég által biztosított BMS – a Texas Instruments-től 3 darab erre a célra kifejlesztett BMS IC-t használ, melyek egyenként 6 cella feszültségének mérésére alkalmasak és 2 hőmérőt is tudnak kezelni, így összesen 18 cella feszültséget és 6 hőmérsékletet tudunk mérni, ami három cellánkként egy hőmérőt jelent. A cellák hőmérsékletének mérése NTC ellenállások segítségével van kivitelezve.

Ahhoz, hogy a BMS más rendszerben is használható legyen, különféle ki- és bemenetekkel kell még az áramkörnek rendelkeznie, melyeket az alábbiakban először felsorolunk, majd ismertetjük funkciójukat:

- DR_EN (Drive Enable) jel:
  Az áramkör képes egy 12V-os kimenettel egy mágneskapcsolót vezérelni, amely az akkumulátor kisütését teszi lehetővé. Rendelkezésünkre áll továbbá egy DR_EN (Drive Enable) bemenet is, amely segítségével több BMS elektronika DR_EN jele felfűzhető és bármelyik meg tudja szakítani a DR_EN jelet.
- CH_EN (Charge Enable) jel:
  A CH_EN a DR_EN-hez hasonló tulajdonságokkal rendelkező jel, azzal a különbséggel, hogy a töltést vezérlő mágneskapcsolót kapcsolja.
- DR_EN_DELAYED jel:
  A DR_EN_DELAYED a DR_EN-hez képest késleltetett jel, hardver szempontjából megegyezik a DR_EN és CH_EN jelek áramkörével.

- AIR_CHK bemenet:

  Az AIR_CHK az akkumulátor izolációs relék (AIR) másodlagos érintkezőit vizsgáló bemenete. Segítségével detektálni lehet, ha valamelyik AIR bekapcsolt állapotban marad. Az akkumulátor izolációs relékre azért van szükségünk, hogy az akkumulátort le tudjuk választani a terhelésről vész esetén. Ennek következménye képpen a terhelés visszakapcsolása előtt szintén szükségünk van a relékre, azért, hogy a terhelés belső kapacitásait először feltöltsük valamilyen áramkorláton keresztül.

A BMS IC-k a mikroprocesszorral SPI-t használva kommunikálnak, a különböző feszültségszintekből adódó probléma miatt pedig egy-egy leválasztó IC-t használunk. A leválasztó IC nagyfeszültségű oldalának tápfeszültségét egy lineáris feszültség referencia szolgáltatja, amely kapcsolható, hogy a BMS kikapcsolt állapotában ne fogyassza az akkumulátor töltöttségét. A lineáris feszültség-referenciát a mikroprocesszor egy optocsatolón keresztül tudja ki-be kapcsolni.

2.1 A BMS szoftveres feladatai:

2.1.1 Működés

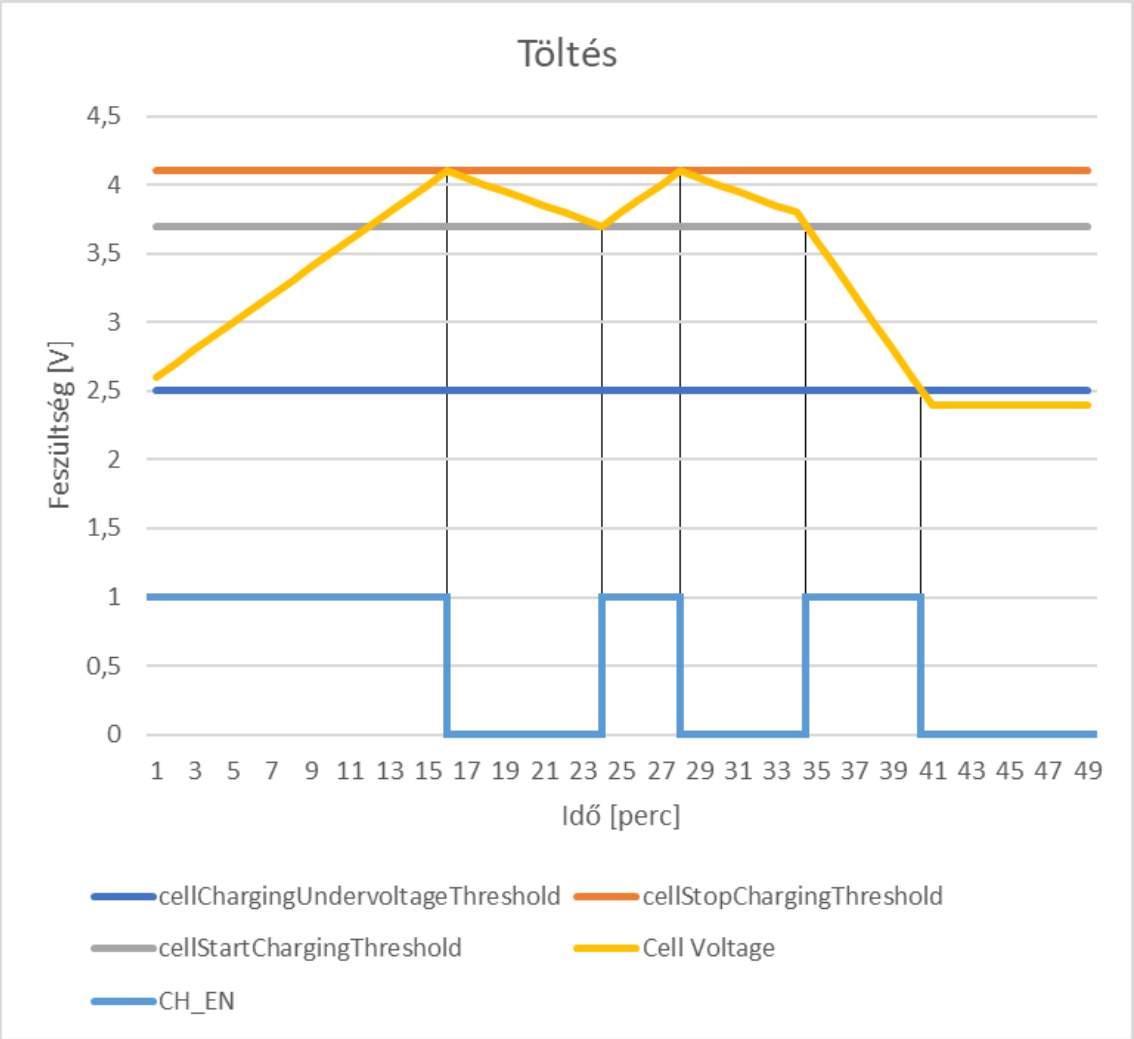
Hajtás engedélyezés (DR_EN)

A hajtást úgy tudjuk aktiválni, hogy az akkumulátor kisütését engedélyezzük, melyet csak bizonyos körülmények között tehetünk meg. Ahhoz tehát hogy ezt megtehessük nem engedhetjük meg, hogy bármelyik cella feszültsége az előírt minimális szint (2.8V) alá essen, továbbá, túl magas cellahőmérséklet, illetve CRC hiba (CRC hiba akkor fordul elő, ha a kommunikáció során a küldött adat meghibásodik) sem fordulhat elő. Ha ezek közül bármelyik fennál, a hajtás tiltásra kerül, melyet a státusz üzenetben is jelzünk. A hajtás tiltását csak az elektronika újraindításával tudjuk megszüntetni.

Töltés engedélyezés (CH_EN)

Az akkumulátor töltéséhez szintén bizonyos feltételeknek teljesülnie kell, a cellák károsodásának elkerülése végett. A cellahőmérők mindegyike 0 és 85 °C közötti (ez a hőmérséklet tartomány cella típusonként erősen változik) hőmérsékletet kell, hogy mutasson, továbbá nem jelezhet az OT – Over Temperature – és az UV – Under Voltage – flag. Az UV flag két esetben is jelezhet: ha túl alacsony a tápfeszültség, illetve, ha valamelyik cella feszültsége túl alacsony. Tehát ezek egyike sem merülhet fel a töltés

engedélyezése esetén. Túlfeszültség egyik cellánál sem fordulhat elő, vagyis nem jelezhet az OV - Over Voltage – flag. Nem fordulhat elő ebben az esetben sem CRC hiba, továbbá minden cellafeszültség a cellChargingUndervoltageThreshold (a cella töltéséhez szükséges minimális feszültségszint) konstans értéke felett, illetve a cellStopChargingThreshold (az a maximális feszültségszint, amely felett már nem tölthető tovább a cella) és a cellStartChargingThreshold (az a maximális feszültségszint, amely alatt a töltés újraengedélyezhető) konstansok értéke alatt kellenek, hogy legyenek. Utóbbi csak akkor szükséges, ha a töltés épp nem engedélyezett. A töltés tiltásra kerül az elektronika újraindításáig, OT, UT, OV és CRC hiba esetén.



2.1. ábra: A CH_EN kimenet működésének illusztrációja egy nem valós töltési folyamat bemutatásával (Vízszintes tengelyen az idő, a függőlegesen pedig a cellafeszültség látható)

A 2.1. ábrán jól látható, hogy a cellChargingUndervoltageThreshold szintet elérve a CH_EN kimenet aktiválódik vagyis a töltés engedélyeződik. Ekkor elkezd töltődni a cella (feltéve, hogy a töltővel össze van kötve), így feszültsége nő. Miután a feszültség

eléri a piros színnel jelölt cellStopChargingThreshold értéket, a töltés befejeződik, vagyis a CH_EN kimenet deaktiválódik. Ekkor a cella lassan merülni kezd, majd miután elérte a szürke színnel jelölt cellStartChargingThreshold értékét, a töltés újraindul. Az ábrán a 36. perctől kezdve jól láthatjuk, hogy a cellafeszültség meredekebben csökken, mellyel azt szimuláljuk, hogy terhelés lett az akkumulátorra kötve. Megfigyelhetjük azt is, hogy a merítés hatására a 42. percben a cellafeszültség 2,5V alá csökken, így e miatt a töltés engedélyezése tiltásra kerül. Tehát ha a cellafeszültség a sötétkék színnel jelölt cellChargingUndervoltageThreshold értéke alatt van, akkor a töltés nem engedélyezett. Ez valójában azért van így, mert egy cella túlmerítése esetén, annak töltésével az oly mértékben károsodhat, hogy a körülötte lévő berendezés is megsérülhet adott esetben.

2.1.2 CAN

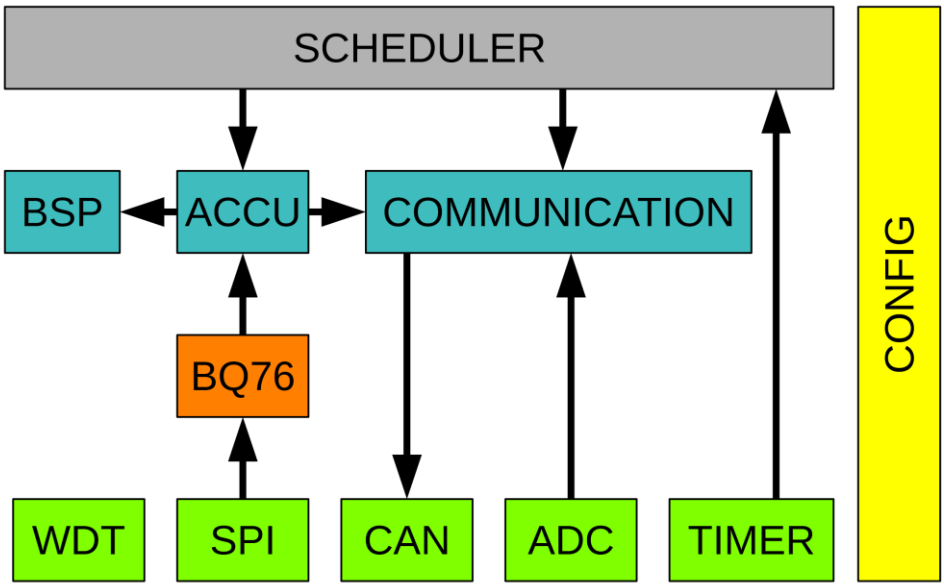
A BMS normál működés esetén alapvetően három-, inicializációs hiba esetén pedig egyedül egy negyedik féle üzenetet küld. Mindegyik üzenet 1Hz-es gyakorisággal kerül kiküldésre és 8byte hosszúak. A 0x300-as ID-vel rendelkező üzenet az első hat cella feszültségét és az első két hőmérő által mért hőmérsékletet tartalmazza, a 0x301-es ID-vel rendelkező üzenet a második hat cella és második két hőmérő, a 0x302-es ID-vel rendelkező pedig a harmadik hat cella feszültségét és a harmadik két hőmérő által mért hőmérsékletet tartalmazza. A 0x340-es ID-vel rendelkező üzenet a státuszt (a szoftver különböző állapotváltozóit tartalmazza), a 0x640-es ID-vel rendelkező üzenet pedig az inicializációs hiba esetén kerül kiküldésre, mely a hibakódokat tartalmazza.

2.2 Szoftver rendszerterv

2.2.1 Szoftver modulok

A szoftver moduláris kialakítású. Alapvetően négy réteget lehet megkülönböztetni:

- driver szint – a mikrovezérlő regiszterszintű vezérlői, egyszerű interfészt biztosítanak a felsőbb rétegek számára
- interfész szint – a drivereket felhasználva egy adott eszköz használatához nyújtanak interfészt
- taszk szint – alkalmazási réteg, a driverek és interfészek használatával hardverfüggetlen módon végzik el a hasznos feladatot
- ütemező szint – a taszkok ütemezéséért felelős



2.2. ábra - Szoftver modulok és kapcsolataik [1]

A 2.2. ábrán látható modulok a következők:

„Config" modul

A szoftver működését meghatározó definíciókat, paramétereket tartalmazó modul. Segítségével könnyedén lehet finom-hangolni a beállításokat.

„Scheduler" modul

A taszkok ütemezéséért felelős. Minden taszk előre definiált időközönként kell lefusson. Egy hardveres timer periféria 1ms-onként megszakítást idéz elő. Minden megszakításkor ellenőrzésre kerül, hogy van-e futtatandó taszk. Ez a modul az új mikroprocesszor használata esetén nem kerül használatra, helyette majd a FreeRTOS operációsrendszer „végzi el" a hozzá tartozó feladatokat.

„BSP" modul

A szoftver hardverfüggő részeit egy úgynevezett „Board Support Package" tartalmazza. Ide tartoznak a különféle mikrovezérlő perifériák konfigurációja, IO lábak vezérlése az akkumulátor állapotától függően stb. Segítségével a hardver megváltozása esetén nem kell az összes többi modulban változtatásokat eszközölni.

„Accu" modul

A BMS IC-kkel való kommunikációért felelős, ez a modul gondoskodik az akkumulátor állapotának felméréséről a „Config" modul paraméterei alapján.

„Communication” modul

A CAN kommunikációért felelős modul. Az „Accu” modul által szolgáltatott adatokat küldi a CAN buszra, a kommunikációs paramétereket a „Config” modul állítja be (pl. üzenetek azonosítója).

„BQ76” modul

A BMS IC-kkel való kommunikáció regiszter szintű modulja, interfész az „Accu” modul és az SPI driver között. Segítségével átlátható módon érhetők el a BMS IC funkciói.

3 A BMS NYÁK-on végrehajtandó módosítások

3.1 A mikroprocesszor cseréjének indoklása

Mivel a megbízó cég már minden termékében STM mikroprocesszorokat használ, ezért célszerű ebben a BMS modulban is kicserélni a régebbi ATMEL processzort. Így a többi termék esetében használt szoftvercsomagok felhasználhatóvá válnak, amelynek köszönhetően a szoftver karbantartása, illetve módosítása és fejlesztése lényegesen leegyszerűsödik. Szintén pozitívum az STM termékek aktívabb támogatása, az STM32CubeMX program elérhetősége, mellyel a processzor perifériáinak inicializálása jóval egyszerűbb, e-mellett az STM úgynevezett HAL (Hardware Abstraction Layer) függvényei sokkal jobban dokumentáltak. Fontos továbbá azt is megjegyezni, hogy az STM nagyobb számítási kapacitása miatt (ezt az 1. táblázat is szemlélteti) lehetőség lesz a későbbiekben bővíteni a BMS funkcióit (pl.: cella belső ellenállás meghatározása, statisztikai adatok gyűjtése).

	ATMEL	**STM**
Adatszélesség	8 bit	32 bit
Frekvencia	16MHz	akár 168MHz
FLASH memória	128K Bájt	akár 1M Bájt
EEPROM	4K Bájt	-
CAN kontroller	1 x CAN interfész	2 x CAN interfész
DMA	-	16 csatornás DMA

3.1. táblázat: Az ATMEL és az STM mikroprocesszor összehasonlítása [2][3]

3.2 Igénybe vett perifériák az ATMEL processzor esetén

A korábban használt ATMEL mikroprocesszort tehát le szeretnénk cserélni egy STM32F4-es eszközre, amely sok mindenben különbözik elődjétől. Más a két processzor lábkiosztása, tokozása, architektúrája, jelszintje, így elkerülhetetlen a NYÁK módosítása. A szoftver BSP modulját átnézve kideríthetjük, hogy a processzor mely perifériáit használjuk, melyet szükséges tudni az STM processzor lábkiosztásának létrehozásához. Ugyanakkor azt is figyelembe kell venni, hogy az ATMEL-es kód több különböző áramkörre íródott, ezért konfigurálható a szoftver és előfordulhat, hogy egyes részek az aktuális hardvernél nincsenek figyelembe véve. Ilyen például az I2C esete, melyre csak

árammérő szenzor jelenlétében van szükség, ez viszont a jelenlegi hardver esetében nincsen.

```
void BSP_Init(void)
{
    BSP_IOInit();
    BSP_ADCInit();
    SPI_MasterInit();
#ifdef CURRENT_SENSOR_AVAILABLE
    I2C_Init(100);
#endif
    BSP_AccuControlInit();
#ifdef CHECK_SUPPLY_VOLTAGE
    // Initialize the supply voltage comparator based watchdog
    AC_Init(SupplyVoltageUndervoltageCb);
#endif
}
```

3.1. ábra: Az ATMEL processzor BSP modul inicializáló függvénye áttekintésével sok használt perifériát megismerhetünk

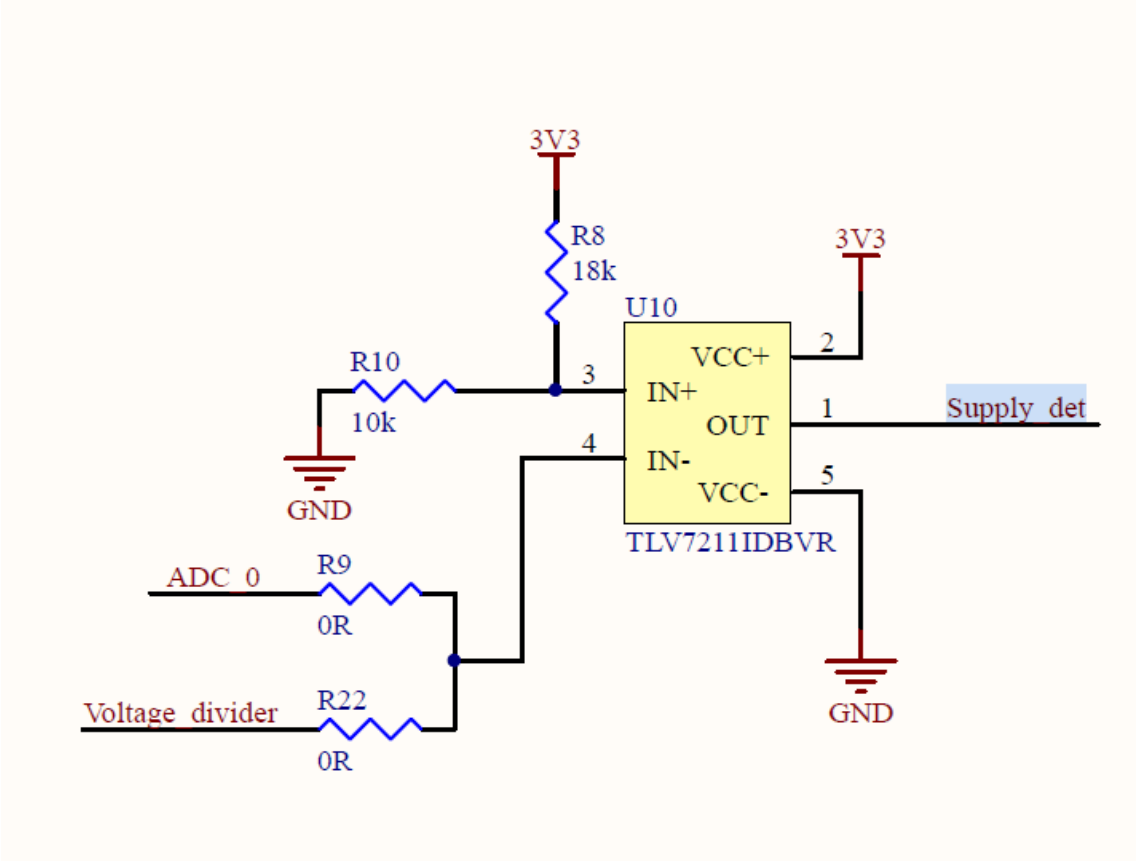
A BSP modult áttekintve az alábbi perifériák használatát találtam szükségesnek:

- Timer (a folyamatok ütemezéséhez kell majd, amely a FreeRTOS operációs rendszer segítségével lesz kiváltva)
- SPI (a BMS IC-kkel való kommunikációhoz)
- 3db GPIO a LED-ekhez
- 3 db GPIO a kimenetvezérléshez (OUT_DR_EN, OUT_CH_EN, OUT_DR_EN_DELAYED)
- 1 db GPIO az SPI izolátor HV (High Voltage) oldali tápjának vezérlésére
- 3 db ADC (táp feszültség, air check, hőmérséklet)
- Analóg komparátor (a tápfeszültség kritikus szint alá csökkenése esetén az időben reagálás végett szükséges)
- CAN (a rendszer többi részével való kommunikációhoz)

3.3 Az STM32-es processzorban használt perifériák:

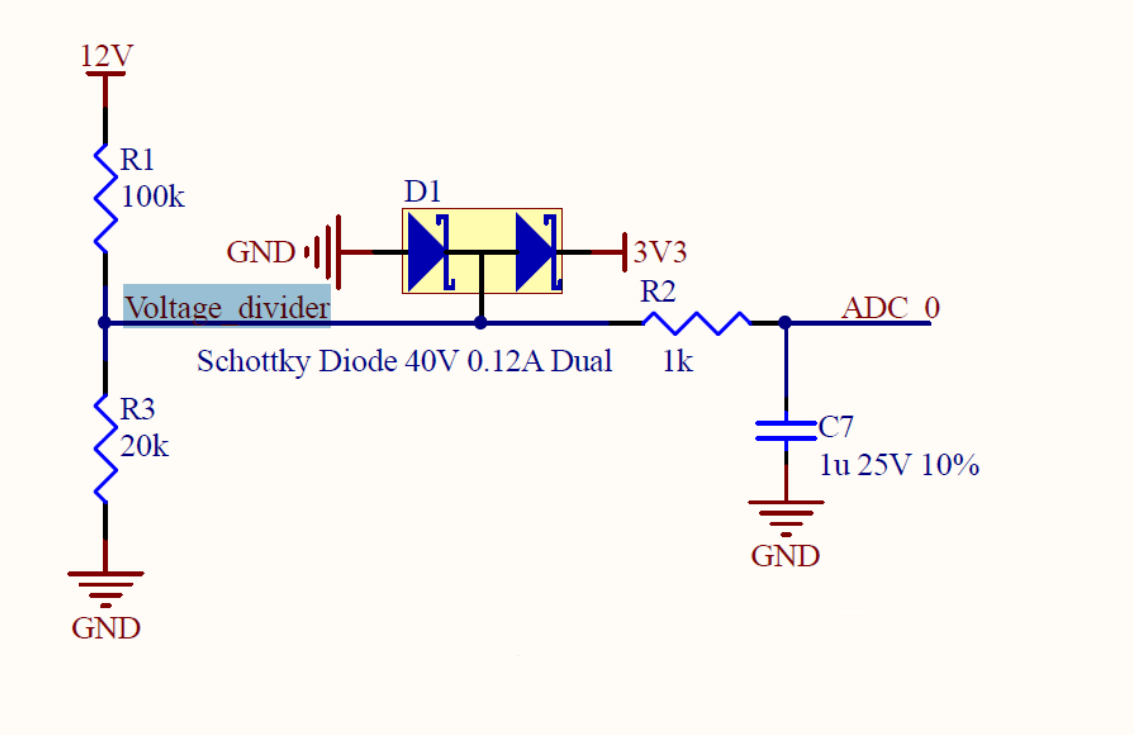
Az előző pontban felsoroltakhoz képest az STM mikroprocesszorban már DMA is használatban lesz, amely az ATMEL processzorban nem állt rendelkezésre. A DMA segítségével, a processzornak nem kell majd foglalkoznia azzal, hogy az ADC csatornák

által beolvasott adatokat az eredmény regiszterből átmásolja egy másik memóriaterületre. Ugyanakkor analóg komparátor sajnos nincs az STM-ben ezért külső komparátor használata szükséges, melynek a kimenete rá lesz kötve a mikroprocesszor egyik lábára, amely megszakítást fog generálni állapotváltozás esetén.



3.2. ábra: Külső komparátor kapcsolási rajza[11]

A 3.2. ábrán látható az ATMEL processzor beépített komparátorát helyettesítő külső komparátor, melynek IN+ bemenetén 1,178V-os referenciafeszültség, IN- bemenetén pedig a bemeneti védett tápfeszültség 1/6-od része jelenik meg.

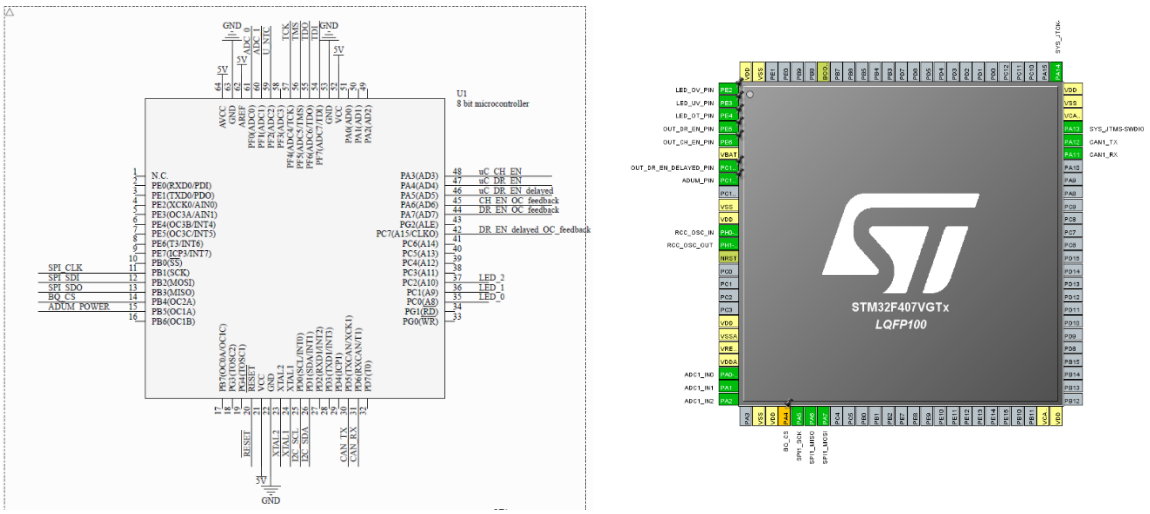


3.3. ábra: Voltage divider jel[11]

Az 1/6-odos osztást – a Voltage divider jelet visszakövetve – a 3.3. ábrán láthatjuk. A 3.2. ábrán megfigyelhető, hogy az R9-es és az R22-es ellenállásokkal kiválasztható, hogy a szűrt, vagy a szűretlen jelet szeretnénk a komparátor IN-bemenetére kötni. A 3.3. ábrán látható két Schottky dióda is, melyek a mikrovezérlő bemenetét hivatottak védeni. A védelem úgy működik, hogy ha a leosztott feszültség 3,3V + 0,4V (a dióda nyitófeszültsége) felé emelkedik, akkor a felső dióda nyit ki, ha pedig a leosztott feszültség GND-0,4V alá esik, akkor az alsó dióda nyit ki, így a mikrovezérlő ADC bemenetére csak -0,4V és 3,7V közötti feszültség kerülhet. A komparátor mindaddig alacsony jelet ad, amíg a tápfeszültség 7,068V felett van. Amint a feszültség ez alá csökken, felfutóél keletkezik, melyet a mikrovezérlő detektálni képes.

3.4 A régi és az új lábkiosztás

Nézzük meg, hogy hogyan néz ki az ATMEL, illetve az STM32 processzorok lábkiosztása:



3.4. ábra: Az ATMEL és az STM32-es mikroprocesszorok lábkiosztásai [4]

Az 2. táblázat bemutatja, hogy az ATMEL mikroprocesszor egyes lábai, milyen néven találhatók meg az STM32 lábkiosztásában. Látható, hogy a táblázat utolsó sorában az STM mikroprocesszornak nincs az ATMEL GPIO_EXTI0 kimenetének megfelelő kimenete, mivel nem rendelkezik beépített komparátorral.

ATMEL	**STM32**
XTAL1	RCC_OSC_IN
XTAL2	RCC_OSC_OUT
LED_0	LED_OV_PIN
LED_1	LED_UV_PIN
LED_2	LED_OT_PIN
uC_CH_EN	OUT_CH_EN_PIN
uC_DR_EN	OUT_DR_EN_PIN
uC_DR_EN_delayed	OUT_DR_EN_DELAYED_PIN
RESET	NRST
ADUM_POWER	ADUM_PIN
AVCC	VDDA
GND (63-as láb)	VSSA
AREF	VREF+

ADC_0	ADC_SUPPLY_VOLTAGE_CH
ADC_1	ADC_AIR_CHECK_CH
U_NTC	ADC_NTC_CH
BQ_CS	BQ_CS
SPI_CLK	SPI1_SCK
SPI_SDI	SPI1_MISO
SPI_SDO	SPI1_MOSI
CAN_TX	CAN_TX
CAN_RX	CAN_RX
TCK	SYS_JTCK-SWCLK
TDI	SYS_JTDI
TDO	SYS_JTDO_SWO
-	GPIO_EXTI0

3.2. táblázat: Az ATMEL és az STM32-es mikroprocesszorok kivezetésének elnevezései

Nem használjuk az ATMEL mikroprocesszor kapcsolási rajzán található következő jeleket (nincsenek bekötve, mivel a szoftver több mást típusú elektronika vezérlésére is szolgál, ahol lehet például I2C periféria is használatban):

- uC_CH_EN_OC_feedback
- uC_DR_EN_OC_feedback
- I2C_SCL
- I2C_SDA

3.5 Jelszintek:

Az ATMEL mikroprocesszor dokumentációjának Electrical Characterictics fejezete (3.3. táblázat) alapján láthatjuk, hogy az egyes ki- és bemenetek milyen jelszinteket produkálnak. Az alábbi táblázat ezt maradéktalanul összefoglalja:

TA = -40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage	Except XTAL1 and RESET pins	– 0.5		0.2 Vcc (1)	V
V_{IL1}	Input Low Voltage	XTAL1 pin - External Clock Selected	– 0.5		0.1 Vcc (1)	V
V_{IL2}	Input Low Voltage	RESET pin	– 0.5		0.2 Vcc (1)	V
V_{IH}	Input High Voltage	Except XTAL1 and RESET pins	0.6 Vcc (2)		Vcc + 0.5	V
V_{IH1}	Input High Voltage	XTAL1 pin - External Clock Selected	0.7 Vcc (2)		Vcc + 0.5	V
V_{IH2}	Input High Voltage	RESET pin	0.85 Vcc (2)		Vcc + 0.5	V
V_{OL}	Output Low Voltage (3) (Ports A, B, C, D, E, F, G)	I_{OL} = 18 mA, V_{CC} = 5V I_{OL} = 8 mA, V_{CC} = 3V			0.7 0.5	V
V_{OH}	Output High Voltage (4) (Ports A, B, C, D, E, F, G)	I_{OH} = – 20 mA, V_{CC} = 5V I_{OH} = – 10 mA, V_{CC} = 3V	4.2 2.4			V

3.3. táblázat: ATMEL mikroprocesszor jelszintjei[3]

Mivel az általunk használt tápfeszültség 5V, ezért a V_{OL} maximum értéke 0.7V, a V_{OH} minimum értéke pedig 4,2V.

A 3.3. táblázatban pedig azt láthatjuk, hogy az STM32-es mikroprocesszornak milyen jelszintjei vannak.

Table 6. Legend/abbreviations used in the pinout table

Name	Abbreviation	Definition
Pin name	Unless otherwise specified in brackets below the pin name, the pin function during and after reset is the same as the actual pin name	
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input / output pin
I/O structure	FT	5 V tolerant I/O
	TTa	3.3 V tolerant I/O directly connected to ADC
	B	Dedicated BOOT0 pin
	RST	Bidirectional reset pin with embedded weak pull-up resistor

3.4 táblázat: Az STM32-es mikroprocesszor jelszintjei [5]

Azt pedig, hogy melyik kivezetés milyen I/O struktúrával rendelkezik (FT, TTa…) a dokumentáció következő oldalán ([5]) kezdődő táblázatból nézhetjük ki.

A 3.5. táblázat összefoglalja a 3.3. és a 3.4. táblázatból következően a két mikroprocesszornál használt jelszintek közötti különbségeket.

ATMEL		STM32	
Kivezetés neve	Jelszint	Kivezetés neve	Jelszint
VCC	5V	VDD	1.8-3.3V
XTAL1	5V	RCC_OSC_IN	-
XTAL2	-	RCC_OSC_OUT	-
LED_0	5V	LED_OV_PIN	3.3V
LED_1	5V	LED_UV_PIN	3.3V
LED_2	5V	LED_OT_PIN	3.3V
uC CH EN	5V	OUT_CH_EN_PIN	3.3V
uC DR EN	5V	OUT_DR_EN_PIN	3.3V
uC DR EN delayed	5V	OUT_DR_EN_DELAYED_PIN	3.3V
RESET	5V	NRST	-
ADUM POWER	5V	ADUM_PIN	3.3V
AVCC	5V	VDDA	1.8-2.4V
GND (63-as láb)	-	VSSA	-
AREF	5V	VREF+	-
ADC_0	0-5V	ADC_SUPPLY_VOLTAGE_CH	0-3.3V
ADC_1	0-5V	ADC_AIR_CHECK_CH	0-3.3V
U NTC	0-5V	ADC_NTC_CH	0-3.3V
BQ CS	5V	BQ_CS	3.3V
SPI CLK	5V	SPI1_SCK	3.3V
SPI SDI	5V	SPI1_MISO	3.3V
SPI SDO	5V	SPI1_MOSI	5V/3.3V
CAN TX	5V	CAN_TX	3.3V
CAN RX	5V	CAN_RX	3.3V
TCK	5V	SYS_JTCK-SWCLK	3.3V
TDI	5V	SYS_JTDI	5V/3.3V
TDO	5V	SYS_JTDO_SWO	3.3V
-	-	GPIO_EXTI0	5V/3.3

3.5. táblázat: Az ATMEL és az STM32-es mikroprocesszor jelszintjeinek összehasonlítása

A 3.5. táblázat alapján mondhatjuk, hogy az STM32-es mikroprocesszor alapvetően 3.3V-os jelszinten kommunikál, tehát kimenetein csak 3.3V-os jel előállítására képes, ugyanakkor a legtöbb láb 5V toleráns, viszont ez csak a bemeneteknél számít, így ezt csak az SPI1_MOSI és a SYS_JTDI lábaknál lehet kihasználni.

Mivel az ATMEL mikroprocesszor 5V-os jelszinten kommunikál, ezért a NYÁK úgy van megépítve, hogy 5V-os jeleket vár a processzortól. E-miatt úgy kellett módosítani a NYÁK-ot, hogy a 3,3V-os jelszintű kommunikáció ne okozzon gondot:

- Az 5V-os LDO-t 3,3V-osra cseréltem a tápban
- Az analóg bemenetek szintillesztőinél az ellenállásosztók újra lettek számolva
- A CAN esetében az IC 3,3V-os tápfeszültségről működőre lett cserélve
- Visszajelző LED-ek előtétellenállásai újra lettek számolva

A fenti felsoroláson kívül felmerült, hogy a BMS IC-k felé használt leválasztó IC-t le kell cserélni az új jelszint miatt, de mivel az működik 3.3V-ról is, ezért ezt nem kellett megtenni. Továbbá a proFET-es teljesítménykimeneteket kapcsoló nFET-eket sem kellett kicserélni, hiszen azok is működnek 3.3V-os szinten.

Nagyon fontos megjegyezni, hogy az új mikroprocesszor jelentősen többet fogyaszt, mint elődje, így célszerű felülvizsgálni a bemeneti kondenzátor kapacitását.

A mikroprocesszorok SPI perifériáját 250kbps-os sebességen működtetjük, így az alvóállapotot jelző üzenet elküldésére legalább 414µs-ot kellett várni. Ahhoz, hogy a BMS chip-eket alvó módba állítsuk, három üzenetet kell elküldeni. Ezek közül az egyik az az üzenet, amely az alvó állapotot előidézi, másik kettő pedig a chip egy bitjét ki, majd bekapcsolja, így egy kimenetet deaktivál, ami enélkül plusz 1mA-t fogyasztana.

Abban az esetben, ha akkor szeretnénk alvó állapotba küldeni a BMS chip-eket, amikor még nem fejeződött be egy ADC konverzió, akkor az alvó állapot csak a konverzió befejezte után következhet be, ami tovább fogyasztja a kondenzátor töltöttségét. Ezt is figyelembe véve az alvó állapot felvétele legfeljebb 1,3ms-ig tart. Tehát, olyan kapacitású kondenzátort kell választani, amely a táp megszűnése esetén, 1,3ms ideig ki tudja szolgálni a mikroprocesszort.

A probléma viszont az, hogy az új mikroprocesszor több áramot fogyaszt, így a bemeneti kondenzátorok még azelőtt lemerülhetnek, hogy alvó állapotba elküldenénk a BMS chip-eket. Az STM32-es processzor adatlapja szerint, a fogyasztás számos tényezőtől függhet, mint például a működési feszültség, a környezeti hőmérséklet, az I/O lábak terhelése, az eszköz szoftver-konfigurációja, a működési frekvencia és a program memóriában elfoglalt helyétől. Az alábbi táblázat bemutatja, hogy az egyes processzor-frekvenciák mellett milyen áram-fogyasztás várható egy demó-kód futtatása esetén:

Symbol	Parameter	Conditions	f_{HCLK}	Typ	Max(2)		Unit
				T_A = 25 °C	T_A = 85 °C	T_A = 105 °C	
I_{DD}	Supply current in Run mode	External clock(3), all peripherals enabled(4)(5)	168 MHz	87	102	109	mA
			144 MHz	67	80	86	
			120 MHz	56	69	75	
			90 MHz	44	56	62	
			60 MHz	30	42	49	
			30 MHz	16	28	35	
			25 MHz	12	24	31	
			16 MHz(6)	9	20	28	
			8 MHz	5	17	24	
			4 MHz	3	15	22	
			2 MHz	2	14	21	
		External clock(3), all peripherals disabled(4)(5)	168 MHz	40	54	61	
			144 MHz	31	43	50	
			120 MHz	26	38	45	
			90 MHz	20	32	39	
			60 MHz	14	26	33	
			30 MHz	8	20	27	
			25 MHz	6	18	25	
			16 MHz(6)	5	16	24	
			8 MHz	3	15	22	
			4 MHz	2	14	21	
			2 MHz	2	14	21	

3.6. táblázat: A processzor áramfogyasztása különböző frekvenciák mellett[3]

Ahhoz, hogy a bemeneti-kondenzátort megfelelő méretűre válasszuk, a processzor áramfogyasztását nem becsülhetjük alul, így a táblázatban azokat a sorokat vesszük figyelembe, amelyekhez az összes periféria bekapcsolása tartozik. E szerint a processzor maximális áramfogyasztása 109mA, mivel 168MHz-es órajelet használunk. A processzor áramfogyasztása mellett figyelembe kell venni a CAN IC-k, a visszajelző LED-ek és a komparátor fogyasztását is. Ismerve az áram-fogyasztást meg tudjuk állapítani, mekkora kapacitású kondenzátorra lesz szükségünk. Mivel a visszajelző LED-ek egyenként 20mA-t, a komparátor 7 µA-t, a CAN IC pedig 17mA-t fogyasztanak, ezért a maximális áramfogyasztás összesen: $109\text{mA} + 20\text{mA} \times 6 + 7\mu\text{A} + 17\text{mA} \cong 246\text{mA}$. Ebből és az alábbi egyenletekből következik a kondenzátor kapacitása.

(1) $\Delta\text{E} = \frac{1}{2}\text{C}(U1^2 - U2^2)$ és

(2) $\text{U1} = 7\text{V}, \text{U2} = 5\text{V}$ és

(3) $\Delta\text{E} = \Delta\text{U} \times \text{I} \times \text{t} = 2\text{V} \times 246\text{mA} \times 1{,}3\text{ms} = 639{,}6\mu\text{J}$,

ezért az (1)-ből kifejezve $C = \frac{2\Delta E}{U1^2\text{-}U2^2} = \frac{2\times 639{,}6\mu J}{49V-25V} = 53{,}3\mu F.$

Szintén fontos megjegyezni, hogy a tápfeszültséget egy aluláteresztő szűrőn keresztül mérjük, mivel a mérésre szukcesszív-approximációs ADC-t használunk, amely egy mintavevő-tartó áramkör segítségével adott ideig mintavételezi a mérendő jelet, majd egy beépített kondenzátor segítségével tartja a feszültséget, amíg meg nem történik az analóg-digitális átalakítás. Ez alatt az idő alatt a mérendő feszültség változhat, így az alvóállapot beállítása később kezdődhet meg a kelleténél. Erre megoldást jelenthet az ADC bemenetének átlagolásához szükséges minták számának csökkentése, viszont szeretnénk megtartani a jövőben a BMS-nek azt a rugalmasságát, hogy szükség esetén pontosabb méréseket tudjunk az ADC-vel elvégezni a minták számának növelésével. E miatt olyan bemeneti kondenzátort (3000μF) választottam, amely még pont belefér a BMS dobozába, így biztosan elegendő töltés áll rendelkezésre az alvó állapot beállításához.

4 A kicserélendő szoftver réteg és a két processzor perifériái közötti különbség bemutatása

Ahhoz, hogy a BMS szoftver megfelelően működjön az új processzorral, ki kell cserélni a driver réteget minden használt periféria esetén. Mivel a két processzor architektúrája különbözik egymástól, ezért előfordulhat, hogy ugyanazon perifériájuk különböző funkciókkal rendelkezik. E miatt először meg kell határozni, hogy az ATMEL processzor egyes perifériái milyen feladatok végrehajtására voltak használva, majd összehasonlítva a két processzor ugyanazon perifériáinak képességeit, létre kell hozni az új driver-t. Az előbbieknek megfelelően a következő pontokban először bemutatásra kerülnek az egyes perifériák által megvalósított funkciók.

A perifériák inicializálását, a GPIO-k konfigurálását és az órajelek beállítását az STM32CubeMX (ezentúl csak CubeMX) program segítségével hajtom végre, amely egy olyan grafikus eszköz, ami az STM mikrokontrollerek és mikroprocesszorok egyszerű konfigurációját teszi lehetővé. A CubeMX program nem csak a kiválasztott perifériák inicializálásához szükséges függvényeket és kódrészeket generálja le, hanem az úgynevezett HAL (Hardware Abstraction Layer) függvényeket is, melyeknek köszönhetően nem kell regiszter-szinten programozni. Fontos továbbá, hogy a FreeRTOS operációs rendszert is a CubeMX segítségével integrálom a kódba, így mindössze annyit kell majd tennem, hogy megadom a task-ok nevét és prioritását, majd a kód generálása után, az egyes task-ok konkrét funkcióját megírom. Nem kell azonban a task-ok létrehozásával időt töltenem.

4.1 CAN

Nézzük meg tehát először, hogy az ATMEL mikroprocesszor CAN perifériája milyen funkciókkal rendelkezik és hasonlítsuk össze az STM32-es mikroprocesszor CAN perifériájával.

4.1.1 Az ATMEL és az STM32-es mikroprocesszorok CAN perifériájának összehasonlítása:

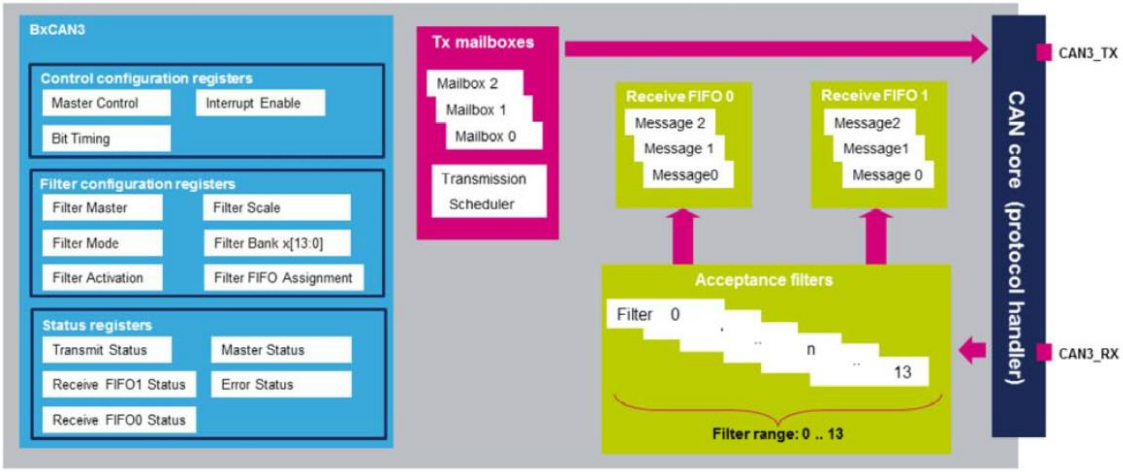
Az ATMEL mikroprocesszor CAN perifériája:

Az ATMEL CAN-je egy úgynevezett full CAN, amiben általában egynél több *Message Object* (továbbiakban MOb) található. Egy MOb az ATMEL esetében egy CAN

keretleíró, amely tartalmaz minden olyan információt, ami egy CAN keret lekezeléséhez szükséges. Konkrétabban minden küldendő, illetve fogadandó üzenet előre definiált a MOb-ban, amely így képes csökkenteni a szoftver terhelését. Ebben az esetben 15 MOb áll rendelkezésre. Új üzenet küldésekor az ATMEL esetén, az üzenetet be kell írni a megfelelő MOb-ba, majd a prioritásának megfelelően kiküldésre kerül a buszra. Minden MOb működtethető *Receive* üzemmódban is, ekkor meg lehet adni egy szűrőt (filter-t), ami alapján a beérkező üzeneteket szűri, így csak az általunk beállított üzenetet vagy üzenet típust engedi fogadni. Az üzenet fogadását követően, ha azt nem tároljuk el más helyre, akkor a következő érkező üzenet felülírja az előzőt, ami így elveszik. Létezik egy *Automatic Respond* mód is, amely úgy működik, hogy a filterrel beállított üzenet vételekor automatikusan, egy általunk beállított választ küld a buszra. Mindezt úgy, hogy eközben nem terheljük a processzort, mivel ez hardveresen van megoldva.

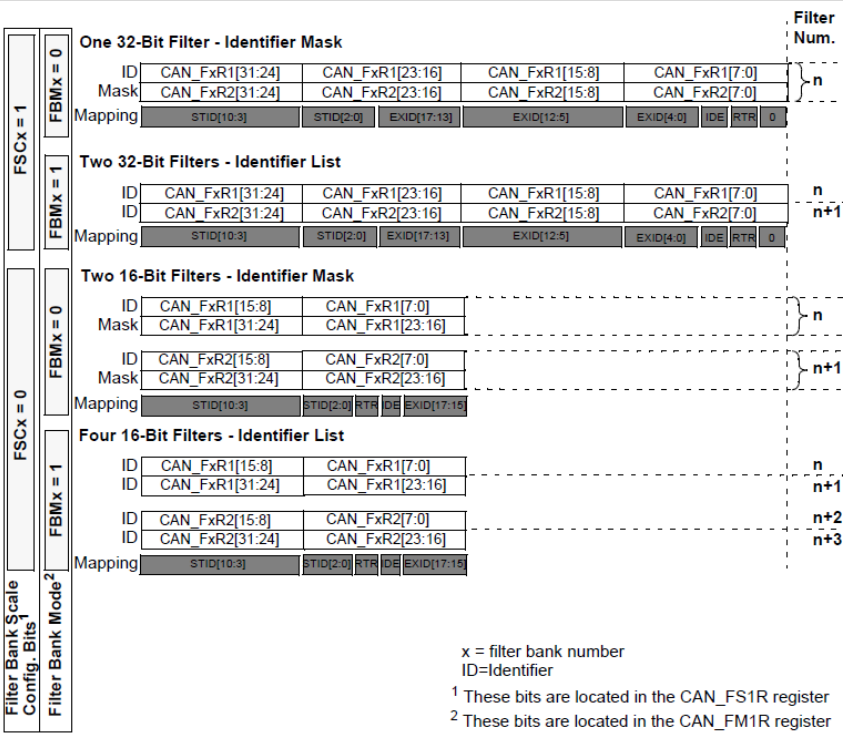
Az STM32-es mikroprocesszor CAN perifériája:

Ezzel szemben az STM32-ben egy úgynevezett *basic extended CAN (bxCAN)* periféria található, amely az üzenetek küldésére három ún. *MailBox*-ot biztosít, üzenetek fogadására pedig két FIFO-t melyek egyenként három üzenet tárolására alkalmasak.



4.1. ábra: Blokkdiagramm-BxCAN [8]

Üzenet küldése esetén prioritás hozzárendeléshez itt is van lehetőség, azonban a kevesebb rendelkezésre álló *MailBox* miatt egyszerre kevesebb üzenet állhat sorban. Üzenetek fogadása esetén itt is van lehetőség szűrőket beállítani, összesen 14 szűrő áll rendelkezésre, melyeket alapvetően két-féle üzemmódban használhatunk, illetve két féle „filter bank scale” beállítására van lehetőség melyet a 4.1. ábra szemléltet.



4.2 ábra: Filter bank "scale" konfigurációja [6]

Egyik üzemmód az „Identifier Mask”, melynek alkalmazása esetén az egyik filter regiszterbe egy ID-t - a másikba pedig a Mask-ot kell írni, így ebben az esetben a Mask-ot tartalmazó regiszterben lévő érték az ID-t tartalmazó regiszter értékét maszkolja, ezáltal az ID azon részei, ahol a Mask 0, nincsenek figyelembe véve. A másik üzemmód az úgynevezett „Identifier List”, melynek esetében pedig mind a kettő filter regiszterben egy-egy ID-t kell írni, így a filter kizárólag erre a két ID-re szűr. Ha kiterjesztett üzenet ID-ket várunk, akkor két 32 bites-, standard üzenet-ID-k használata esetén pedig négy 16 bites filter regiszter áll rendelkezésre. Utóbbi esetében „Identifier Mask” üzemmódban egy-egy ID és Mask regiszter-, „Identifier List” üzemmódban pedig 4 darab ID regiszter-használható.

4.1.2 Az ATMEL CAN driverének bemutatása

Az ATMEL processzor CAN drivere a következő függvényekből áll:

- `void can_init( CAN_Baudrates baud)`

  Ezzel a függvénnyel beállítjuk a bitidőt és engedélyezzük a CAN perifériát

- `bool can_tx( char mob,  CAN_packet *packet)`

Ezzel a függvénnyel tudunk üzenetet küldeni a buszra, úgy, hogy megadjuk, hogy melyik MOb-ból szeretnénk azt elküldeni, amellyel ugyanakkor a prioritást is beállíthatjuk.

- `bool prepare_rx( char mob, uint32_t id, uint32_t idmask, CAN_cbf callback)`

  Ezzel a függvénnyel az általunk kiválasztott MOb-ot beállíthatjuk úgy, hogy a függvény paramétereiként megadott adatok alapján szűrje a beérkező üzeneteket, üzenet érkezése esetén pedig a megadott Callback függvény fusson le.

- `inline void wait_until_mob_valid(char mob)`

  Ennek a függvénynek a segítségével addig várhatunk, amíg a paraméterként megadott MOb használatban van.

4.1.3 Az STM32 CAN driverének bemutatása

MX_CAN1_Init

A periféria inicializálását az **`MX_CAN1_Init`** függvény végzi, melyet a CubeMX program segítségével generáltam. Eredetileg a függvény nem várt paramétert, de kiegészítettem úgy, hogy meg lehessen adni, hogy milyen baudrate-et szeretnénk beállítani, amely az ATMEL-es kód esetében is így volt.

can_tx

Az üzenetek küldésére a **`can_tx`** függvényt írtam meg, melynek az ATMEL-es kódban lévő megfelelője ugyanilyen névvel rendelkezik, viszont a két processzor CAN perifériájának funkcionális és architektúrális különbségeiből adódóan az a programkód itt egyáltalán nem működne ezért teljesen újra kellett gondolni. Paraméterként a függvénynek meg kell adni a küldeni kívánt üzenetet, amely struktúraként van átadva.

```
bool can_tx(CAN_packet *packet)
{
	CAN_TxHeaderTypeDef txHeader;
	uint32_t txMailBox;

	#ifdef CAN_2_0_B
	assert(packet->id <= 0x1fffffff);
	#else
	assert(packet->id <= 0x7ff);
	#endif
	assert(packet->length <=8);

	txHeader.DLC = packet->length;
	txHeader.StdId = packet->id;
	txHeader.IDE = packet->ideBit;
```

```
	txHeader.RTR = CAN_RTR_DATA;

	if(HAL_CAN_AddTxMessage(&hcan1, &txHeader, packet->data, &txMailBox)
	!= HAL_OK){
		Error_Handler();
		return false;
	}
	return true;
}
```

A függvény először megvizsgálja, hogy a küldendő üzenet ID-je a használt CAN üzemmódnak megfelelő formátumú-e, majd leellenőrzi, hogy az üzenet adat része megfelelő méretű-e. Ezután feltölti a *txHeader* struktúrát a paraméterként megadott struktúra elemeivel majd feltölti az üzenetet az egyik üres *MailBox*-ba a **HAL_CAN_AddTxMessage** függvény segítségével, melynek visszatérési értékétől függően, igaz vagy hamis értékkel tér vissza a **can_tx** függvény. Az ATMEL-es kódtól eltérően itt nem kell átadni a függvény paraméterlistájában a használni kívánt *MailBox* számát, hiszen a **HAL_CAN_AddTxMessage** függvény a küldendő üzenetet a legelső szabad *MailBox*-ba tölti fel.

prepare_rx

Az üzenetek fogadására a **prepare_rx** függvényt írtam meg, melyet szintén teljesen újra kellett gondolni. Paraméterként a függvénynek meg kell adni a fogadó FIFO számát (0 vagy 1), egy ID-t és egy mask-ot – a filternek – és egy függvénypointert, amely arra a függvényre mutat, melyet az üzenet beérkezésekor kell lefuttatni.

```
bool prepare_rx(uint32_t ReceiveFIFO, uint32_t id, uint32_t idMask, CAN_cbf
callback){
	CAN_FilterTypeDef filter0;
	uint32_t idHigh, idLow, idMaskHigh, idMaskLow;

	assert_param(IS_CAN_FILTER_FIFO(ReceiveFIFO));
	canlist[(unsigned)ReceiveFIFO] = callback;
	idHigh = 16 >> id;
	idLow = id & 0x0000ffff;
	idMaskHigh = 16 >> idMask;
	idMaskLow = idMask & 0x0000ffff;

	filter0.FilterBank = ReceiveFIFO;
	filter0.FilterFIFOAssignment = ReceiveFIFO;
	filter0.FilterMode = CAN_FILTERMODE_IDMASK;

#ifdef CAN_2_0_B
	filter0.FilterScale = CAN_FILTERSCALE_32BIT;
#else
	filter0.FilterScale = CAN_FILTERSCALE_16BIT;
#endif
	filter0.FilterIdHigh = idHigh;
```

```
        filter0.FilterIdLow = idLow;
        filter0.FilterMaskIdHigh = idMaskHigh;
        filter0.FilterMaskIdLow = idMaskLow;
        filter0.FilterActivation = CAN_FILTER_ENABLE;

        HAL_CAN_ConfigFilter(&hcan1, &filter0);

        HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING ||
        CAN_IT_RX_FIFO1_MSG_PENDING);

        return true;
}
```

A függvény először leellenőrzi, hogy a paraméterként megadott FIFO azonosító helyes értékű-e, majd a megkapott Callback függvény-pointert elmenti egy globális tömbbe. Ezután felkonfigurálja a filtert, a **`HAL_CAN_ConfigFilter`** függvény segítségével. Ehhez viszont szükség van egy `CAN_FilterTypeDef` struktúrára, melynek az elemeinek az értékét, a paraméterként megkapott adatok és korábbi megfontolások alapján választottam meg, melyeket a következő pontokban részletezem.

A `CAN_FilterTypeDef` struktúra `FilterBank` elemének a paraméterként kapott `ReceiveFIFO` változót adtam értékül, amellyel beállítottam a használni kívánt FIFO-t.

A `FilterFIFOAssignment` struktúraelemmel lehet kiválasztani a használni kívánt filtert, melynek szintén a `ReceiveFIFO` változót adtam értékül. Ebből következik, hogy legfeljebb két filter használható, tehát a szoftver két különböző üzenetcsoport érkezését várhatja.

A `FilterMode` struktúraelem értékének megválasztásával lehet beállítani a filter-üzemmódot, amely jelen esetben – a függvény paraméterlistájából következően – „identifier mask” kell, hogy legyen, így a filter is egy ID-t és egy mask-ot fog várni.

A `FilterScale` struktúraelemmel lehet beállítani a filter-regiszterek méretét, amely attól függ, hogy milyen típusú üzenetet várunk. A várt üzenet lehet kiterjesztett (extended) - melynek esetén 32 bites filter regiszterekre van szükség -, vagy pedig standart méretű, ez esetbe elég a 16 bites filter regiszter.

A `FilterIdHigh` és `FilterIdLow` elemeknek – az „identifier mask” üzemmód használata esetén – az ID felső, illetve alsó 16 bitjét kell megadni. Hasonlóan, a `FilterMaskIdHigh` és `FilterMaskIdLow` elemeknek a mask felső, illetve alsó 16 bitjét kell megadni.

A `FilterActivation` struktúraelemmel lehet az adott filtert engedélyezni, illetve tiltani. Mivel használni szeretnénk a filtert, ezért természetesen engedélyezzük azt.

Az egyes struktúraelemek értékeinek megadását követően-, ahhoz, hogy az általuk képviselt beállítások érvényre jussanak, meg kell hívni a **HAL_CAN_ConfigFilter** függvényt, melynek paraméterként át kell adni a struktúrát.

Ezután aktiváljuk a megfelelő értesítéseket a **HAL_CAN_ActivateNotification** függvényt meghívva, hogy a felkonfigurált filteren keresztüli üzenet érkezése esetén megszakítás történjen. Fontos szempont, hogy attól függően, hogy melyik filteren keresztül érkezett az üzenet, más és más megszakításkezelő függvény kell, hogy lefusson, hiszen a különböző üzenetekhez, más Callback függvény tartozik, melyet a megszakításkezelő rutinból hívunk meg. Az STM32-es processzor CAN perifériájának felépítéséből adódóan – az ATMEL processzorral ellentétben – nincsenek olyan flag-ek, melyek egy bizonyos filteren keresztül érkező üzenet beérkezését jeleznék, ezért nem is tudunk olyan megszakítást aktiválni, ami ebben az esetben futna le. Olyan megszakítást viszont lehet aktiválni, amely az egyik FIFO-ba érkező üzenet esetén fut le. Aktiválva tehát a két FIFO-hoz tartozó megszakítást, üzenet érkezése esetén lefut a **HAL_CAN_RxFifoxMsgPendingCallback** megszakításkezelő függvény, amely kiolvassa az adott FIFO-ból a beérkező üzenetet és eltárolja egy helyi változóba, majd meghívja a megfelelő Callback függvényt. A Callback függvény paraméter listáját úgy módosítottam, hogy egy `CAN_RxHeaderTypeDef` struktúrát és az üzenetre mutató pointert várjon egy *CAN_packet* helyett, így nem kell időt pazarolni arra, hogy a `CAN_RxHeaderTypeDef` struktúrából és az üzenet-pointerből előállítsunk egy `CAN_packet` struktúrát mert azt a függvény törzsében úgyis újra szét kéne bontani.

HAL_CAN_RxFifoxMsgPendingCallback

Ez a függvény akkor fut le, ha üzenetet sikerült fogadni az egyik FIFO-ba, így ezt arra használjuk fel, hogy az üzenet érkezésekor a megfelelő Callback függvényt meghívjuk. Ehhez először kiolvassuk a FIFO-ból a beérkező üzenetet a **HAL_CAN_GetRxMessage** függvény segítségével, majd attól függően, hogy melyik FIFO-ba érkezett üzenet, meghívjuk az ehhez tartozó Callback függvényt. Ha a **HAL_CAN_RxFifo0MsgPendingCallback** függvény futott le, akkor a Callback függvényeket tartalmazó tömb nulladik eleme, ellenkező esetben pedig az első eleme fut le.

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan) {
      /* Prevent unused argument(s) compilation warning */
        UNUSED(hcan);

        CAN_RxHeaderTypeDef rxHeader;
        uint8_t message[64];
```

```
        HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rxHeader, message);
        /*
         * Call Back függvény meghívása
         */
        canlist[CAN_RX_FIFO0](&rxHeader, message);
}
```

4.2 ADC

Az ADC perifériát három feszültség mérése végett kell használni, amely az ATMEL mikroprocesszor esetén még DMA nélkül volt használva. Mivel az STM32-ben már elérhető a DMA, ezért mindenképpen célszerű használni, így nem kell minden egyes ADC konverzió esetén megszakítani a CPU munkamenetét ahhoz, hogy elmentsük a beolvasott adatot.

4.2.1 ADC driver bemutatása

Az ADC inicializálása

Az ADC inicializálásához a main függvényből meghívom a **`BSP_ADCInit`** függvényt, amely értéket ad az `adcData` struktúráknak majd meghívja a **`MX_ADC1_Init`** függvényt, ami az inicializálást végzi. Ezt a függvényt az STM32CubeMx generálta, az általam a programban kiválasztott funkciók figyelembevételével, melyek a következők: 3db ADC csatorna aktiválasztása, 10 bites felbontás, scan konverzió, folyamatos konverzió, folytonos DMA kérés kiválasztása és a konverziók számának 3-ra állítása. Ezután a **`ADC_StartCyclicConversion`** függvény meghívásával elindítom az ADC konverziót.

ADC_StartCyclicConversion

Az ADC konverzió elindítását végzi ez a függvény, paramétert nem vár, visszatérési értéke nincsen. A függvény törzsében meghívjuk a **`HAL_ADC_Start_DMA`** függvényt, melynek paraméterként meg kell adni egy `ADC_HandleTypeDef` struktúrát, ami tartalmazza az ADC periféria beállításait, egy pointert, ami az eredménybuffer helyére mutat, és az eredmény hosszát bitben jelző értéket. Ennek a függvénynek a lefutását követően minden beolvasott érték a kijelölt memóriaterületen megtalálható lesz.

HAL_ADC_ConvCpltCallback

Ez a függvény akkor hívódik meg, amikor befejeződik egy konverzió. Ekkor elvégzendő feladat az `adcData` struktúrák `result_raw` tagjuk értékének frissítése, amik az ADC csatornák által érzékelt feszültséget tárolják. Ezután meg kell vizsgálni, hogy be van-e állítva a folyamatos konverzió, mert ha nem, akkor az átalakítást le kell állítani. Az ADC

konverziók leállításához a `HAL_ADC_Stop_DMA` függvényt használom, mivel az elindításhoz a `HAL_ADC_Start_DMA` függvényt használtam.

ADC_GetVoltage

Ezzel a függvénnyel tudom lekérdezni az egyes ADC-csatornákra kapcsolt feszültség értékét. Paraméterként a leolvasni kívánt csatorna sorszámát kell megadni. A függvény törzsében először leellenőrzöm, hogy a megadott paraméter 0 és 2 között van-e, ha nem, akkor -1-gyel tér vissza (mivel -1V-os érték nem fordulhat elő ezért ez egy jó választás). Ezt követően lekérdezem a kiválasztott ADC csatornán beolvasott nyers adatot és megszorzom a hozzátartozó szorzóval, mindezt egy kritikus szakasz keretében, hogy a nyers adaton végzett művelet közben esetlegesen érkező megszakítás ne tegye tönkre az eredményt.

Mivel FreeRTOS operációs rendszert használunk, ezért a kritikus szakasz nyitását az `_enterCritical` függvénnyel, bezárását pedig az `_endCritical` függvény meghívásával tudom megtenni. A kritikus szakaszt érdemes a lehető legrövidebbre írni, hogy ne tartsa fel a processzort. E-miatt itt az ADC adott csatornájának nyers értékét egy ideiglenes változóba elmentem, majd kilépve a kritikus szakaszból, az ideiglenes változón végzem el a szükséges műveletet.

4.3 FreeRTOS

A Timer-perifériát az egyes task-ok ütemezésére használtuk a régi hardveren futó szoftverben. Az új processzor viszont már lehetőséget nyújt FreeRTOS operációs rendszer használatára, amelyet kihasználva nem lesz szükség a Timer-perifériára. Az operációs rendszert a CubeMX program segítségével integráltam a szoftverbe, azáltal, hogy – a programon belül a FreeRTOS-ra rákattintva – az interface-t CMSIS_V1-re állítottam, ezzel aktiválva az operációs rendszert, majd a Tasks and Queues fül alatt hozzáadtam a szükséges task-okat, hogy a kódban ne kelljen majd létrehozni őket. Négy task-ot hoztam így létre, melyek a következők: AccuTask, StatusTask, InitErrorTask és PrechargeTask. Az AccuTask nevű task azokat a műveleteket hajtja végre, melyek a régi szoftverben az accuTaskTimer Timer-struktúra lejárta esetén futottak le. Hasonlóan a többi task esetén is, a StatusTask nevű task a statusMessageTimer Timer-struktúra-, az InitErrorTask nevű task a initerrorMessageTimer Timer-struktúra-, a PrechargeTask pedig a prechargeTimer Timer-struktúra lejárta esetén lefutó műveleteket hajtja végre. Az egyes task-ok prioritását a hozzájuk tartozó Timer-struktúrákban tárolt időzítési

értékek szerint állítottam be. A kis időközönként lefuttatott task-oknak magasabb-, a nagyobb időközönként lefuttatott task-oknak pedig alacsonyabb prioritást állítottam be. Az accuTaskTimer 20-, a statusMessageTimer 200-, a initerrorMessageTimer 200-, a prechargeTimer Timer struktúrában pedig 4000 ms-ra volt beállítva az időzítés, így az AccuTask-nak van a legmagasabb, a StatusTask-nak és az InitErrorTask-nak közepes, a PrechargeTask-nak pedig a legalacsonyabb a prioritása.

4.4 SPI

Az SPI periféria a BMS chip-ekkel való kommunikáció végett szükséges, amely elengedhetetlen a cellák állapotának megismeréséhez. A perifériát használó függvények a BQ76.c fájlban találhatók, melyek közül az alább bemutatott függvényeket kellett újraírnom.

4.4.1 Az SPI driver bemutatása:

BQ76_RegWrite

Ennek a függvénynek a segítségével lehet egy egy-bájtos adatot írni az adott eszköz adott regiszterébe. Paraméterként az eszközcímet (amellyel kommunikálni szeretnénk), az eszköz egy bizonyos regiszterének a címét és a küldendő adatbájtot kell megadni. A függvény törzsében először feltöltöm a `crcHelper` tömböt a paraméterlistában megadott adatokkal (ez lesz majd az elküldendő üzenet), majd kiszámolom a **`BQ76_CheckCRC`** függvény segítségével az ehhez tartozó ellenőrző összeget. Ezután a Chip Select lábat alacsony szintűre állítom, ezzel engedélyezve az eszközt, majd elküldöm a `crcHelper` tömb tartalmát, végül pedig az ellenőrző összeg értékét és a Chip Select lábat magas szintűre állítom, így letiltva az eszközt.

BQ76_RegReadRaw

Ezzel a függvénnyel adott eszköz adott regiszteréből adott hosszúságú adatot lehet kiolvasni. Paraméterként az eszköz címét, az eszköz olvasni kívánt regiszterének a címét, az olvasni kívánt adat hosszát és a beolvasott adatot tároló tömb címét kell megadni. A függvény törzsében először feltöltöm a `crcHelp` tömböt az olvasni kívánt eszköz címével, az olvasni kívánt regiszter címével és az olvasni kívánt adat hosszával, majd a Chip Select lábat alacsony szintre állítom, ezzel engedélyezve az eszközt. Ezután elküldöm a `crcHelper` tömb elemeit a **`HAL_SPI_Transmit`** függvény segítségével, majd a **`HAL_SPI_Receive`** függvény segítségével beolvasom a kívánt mennyiségű adatot, ezután

pedig beolvasok még egy adatot, amely a CRC érték lesz. Ezt követően a Chip Select lábat magas értékűre állítom, majd leellenörzöm a CRC értékét a `BQ76_CheckCRC` függvény segítségével.

4.5 EEPROM

Az EEPROM olyan tulajdonsággal rendelkezik, hogy a benne tárolt adatok a tápfeszültség kimaradása esetén is megmaradnak, ezért érdemes a fontos változókat itt tárolni.

Az ATMEL mikroprocesszor esetében mindössze 4Kbyte EEPROM-mal gazdálkodtunk, ezzel szemben az STM32-es mikroprocesszorban nincs EEPROM, csak Flash memória, amely 12 szektorból áll melyek összesen akár 1Mbyte-os férőhelyet is biztosíthatnak.

Table 5. Flash module organization (STM32F40x and STM32F41x)

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	. . .	. . .	. . .
	Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

4.1 táblázat: Az STM32-es mikroprocesszor Flash modulja [9]

Ebből következik, hogy a 12 szektorból nekünk mindössze 1-re van szükségünk. Az emulált EEPROM számára a 11. szektort választottam, azért, hogy más szoftver-rétegek – melyek a 0. szektortól kezdik feltölteni a flash-t – futás közben ne írják felül az itt tárolt adatok értékét. Ehhez az STM32F407VG_FLASH.ld fájlban a Flash memóriához tartozó hosszértéket 896 Kbyte-ra állítottam, így a felhasználói program a 11. szektort már nem használhatja.

4.5.1 Az EEPROM driver bemutatása

Az ATMEL mikroprocesszorra írt driver az alábbi két függvényt tartalmazza:

- `void EEPROM_Write(const uint16_t address, const uint8_t* data, const uint8_t length);`

  Ez a függvény az adott memóriacímre adott méretű adatot ír. Paraméterként meg kell adni neki az írás helyének a címét, az írni kívánt adatra mutató pointert és az írni kívánt adat hosszát.

- `void EEPROM_Read(const uint16_t address, uint8_t* data, const uint8_t length);`

  Ez a függvény az adott memóriacímen lévő adott hosszúságú adatot kiolvassa az általunk megadott helyre. Paraméterként meg kell adni az olvasás címét, a kiolvasott adatot tároló hely címét és a kiolvasott adat hosszát.

Az STM32-es mikroprocesszor driverében ugyanilyen néven megtalálhatók ezek a függvények, melyeknek a paraméterlistájuk is azonos, viszont a függvény-törzsük teljesen mást tartalmaznak az ATMEL-es megfelelőjükhöz képest. Nézzük meg tehát a következő pontokban, hogy ezeknek a függvényeknek a törzse mit tartalmaz.

EEPROM_Write

A függvény törzsében először előállítom az abszolút memóriacímet, úgy, hogy a 11. memóriaszektor kezdőcíméhez hozzáadom a paraméterben megadott relatív cím értékét. Ezt követően, feloldom a Flash-t a **`HAL_FLASH_Unlock`** függvénnyel, majd egy for ciklusban beírom a megadott memóriacímre a megadott adatot a **`HAL_FLASH_Program`** függvény segítségével, melynek a `TypeProgram` paraméterének a `FLASH_TYPEPROGRAM_BYTE` értéket adom, így az adat írása bájtonként történik. Miután a for ciklus a végéhez ért, a **`HAL_FLASH_Lock`** függvénnyel lezárom a Flash-t.

EEPROM_Read

A függvény törzsében itt is először előállítom az abszolút memóriacímet, ugyanúgy, mint az **`EEPROM_Write`** függvény elején, majd egy for ciklusban elvégzem az olvasást, amit úgy teszek, hogy a paraméterlistában megadott `readBuf` pointer által mutatott memóriacímen lévő adatot felülírom a Flash aktuális címén lévő adatával.

4.6 Watchdog

A Watchdog periféria a szoftver működésében fellépő hibák jelzésére és feloldására szolgál, melyet a szoftver újraindításával old meg. Az STM32-es mikroprocesszorban két féle Watchdog található: egy ún. Independent Watchdog (IWDG) és egy Window Watchdog (WWDG). Mindkettő ugyanazt a célt szolgálja, viszont az IWDG-t olyan alkalmazásokban ajánlott használni, ahol a fő alkalmazástól teljesen független folyamatként kell futnia, de alacsonyabb időzítés pontossági korlát van előírva. A WWDG pedig olyan alkalmazásokban használandó, ahol megkövetelendő, hogy a watchdog egy pontos időkereten belül reagáljon.

Én az IWDG-t használom, mivel nincs semmilyen követelmény arra vonatkozóan, hogy a watchdog milyen pontossággal dolgozzon. A watchdog órajelforrása a 32kHz-es LSI, melyet egy 4-es órajel osztó 8kHz-re állít. Így például, ha 1s-os időt szeretnénk beállítani, akkor a watchdog-nak 8000-et kell számolnia, tehát a maximális felbontás 0.125ms, ami bőven elég, tekintve, hogy a szoftverben a watchdog-nak beállított minimális idő 15ms.

4.6.1 A watchdog driver bemutatása

MX_IWDG_Init

A watchdog driver fontos része az inicializáló függvény, amely beállítja a megfelelő prescale szorzót és egy alapértelmezett értéket betölt a *IWDG_RLR* regiszterbe. Ezt a függvényt a CubeMX program generálta le, melyben a prescale értékét 4-re, a reload value értékét pedig az alapértelmezett 4095-re állítottam. Az utóbbi értéket szoftverből változtatni fogjuk, ezért nem fontos, hogy a periféria inicializálásakor mire állítjuk.

IWDG_Start

Ezzel a függvénnyel lehet, a watchdog időzítőjét átállítani. A függvény törzsében először engedélyezem a periféria védett regisztereinek az írását a `IWDG_ENABLE_WRITE_ACCESS` függvény meghívásával, amely a *IWDG_KR* regiszterbe beírja a 0x5555 értéket. Ezután beírom a paraméterként megadott értéket a *IWDG_RLR* regiszterbe, majd a `__HAL_IWDG_RELOAD_COUNTER` függvényt meghívva a 0xAAAA értéket a *IWDG_KR* regiszterbe, így a *IWDG_RLR* regiszterbe az általam beírt érték átkerül a watchdog időzítőjébe. Végül az `IWDG_DISABLE_WRITE_ACCESS` függvényt meghívva tiltom a periféria védett regisztereinek írását.

IWDG_reset

Ezt a függvényt meghívva az IWDG számlálót újra feltöltöm a *IWDG_RLR* regiszterben lévő értékkel. A függvény törzsében ezt a funkcionalitást a `HAL_IWDG_Refresh` függvény meghívásával érem el, melynek visszatérési értékét vizsgálva tudom, hogy sikerült-e lefutnia. Ha nem, meghívom az `Error_Handler` hibakezelő függvényt.

5 A BMS tesztelése

A teszteléshez szükséges egy akkumulátor modul, amely 12 vagy 18 darab cellát tartalmaz. Első körben a BMS cellakiegyenlítő funkciójának tesztelését mellőztem, hogy láthassam, valós értékeket kommunikálnak-e a BMS IC-k a mikroprocesszor felé. Ahhoz, hogy a kommunikált értékeket kényelmesen ellenőrizhetni lehessen, 12, illetve 18 darab teljesítmény-ellenállást sorba kötöttem, majd a kivezetéseit bekötöttem a BMS áramkörbe, így a feszültségértékek ellenőrzésekor egy multiméterrel könnyedén hozzáférhettem a megfelelő mérési pontokhoz.

Amikor elindítottam a programkódot, a StartAccuTask nevű taszk AccuInit függvénye hibával tért vissza. Debug módban kiderítettem, a hiba abból adódik, hogy az AccuInit `BMSICInit` függvénye nem képes hiba nélkül lefutni. Ez a függvény a BMS IC-k és a processzor közötti kommunikáció létesítéséért felelős, ami annyit jelent, hogy leellenőrzi, hogy valóban jelen van-e annyi BMS IC, amennyi a config.h fájlban van feltüntetve, majd ennek megfelelően inicializálja őket. A hiba miatt a függvény egy BMS IC-vel sem tudott kapcsolatba lépni, az SPI vonalra kiküldött üzenetekre rendre nem érkezett válasz. Ahhoz, hogy kiderítsem, hogy mi lehetett a probléma, először leellenőriztem – oszcilloszkóp segítségével – hogy a mikroprocesszor valóban azt az adatot küldi, mint amit a programban beállítottam. Ezt követően megnéztem, hogy a NYÁK nagyfeszültségű oldalán megjelenik-e helyesen a kiküldött adat. Kiderült, hogy a Chip Select jel egyáltalán nem reagált, végig alacsony szintű volt, ráadásul ez a jel a kisfeszültségű oldalon is folyamatosan 0 volt. Miután leellenőriztem, hogy helyes lábkiosztás van-e beállítva a Chip Select jelhez, észrevettem, hogy a kiosztás ugyan helyes, a jel GPIO módjához viszont a `GPIO_MODE_AF_PP` makró volt beállítva, ami a jel hardveres előállítása esetén használandó. A Chip Select jelet ez esetben pedig a szoftver állítja, így a korábbi makrót átírtam a megfelelőre (`GPIO_MODE_OUTPUT_PP`). Így lefuttatva a programot a Chip Select jel megfelelően működött mind a kis- és a nagyfeszültségű oldalon. Azonban-, a jel útjában volt még egy negálás, így fordított logikával működött. E miatt a MISO (Master In Slave Out) jel folyamatosan alacsony szinten volt mivel a Chip Select jellel nem sikerült aktiválni az IC-t, így hiába jelent meg az üzenet a MOSI vonalon, az IC ezt nem fogadta, ezért nem is válaszolt. Miután ezt a programkódban javítottam az SPI vonalon a megfelelő adatfolyamot láthattam.

A Chip Select jel javítását követően a BMS IC-k inicializálásáért felelős függvény hiba nélkül lefutott. Ezt követően leellenőriztem, hogy a BMS IC-k megfelelő értékeket kommunikálnak-e, amit úgy tettem meg, hogy miután hagytam egy ideig futni a programot, megállítottam és megnéztem a `systemValues.cellVoltages` tömb elemeinek az értékét és összehasonlítottam a cellákat szimuláló ellenállásokon eső feszültségértékkel. A `systemValues.temperatures` tömb értékei esetén pedig megnéztem, hogy a kezdeti értékük megegyezik-e a szoba hőmérsékletével, illetve, hogy ha a kezemmel melegítem a hőmérőt, változik-e az érték. A feszültségértékek és a cellahőmérsékletek is helyesnek bizonyultak.

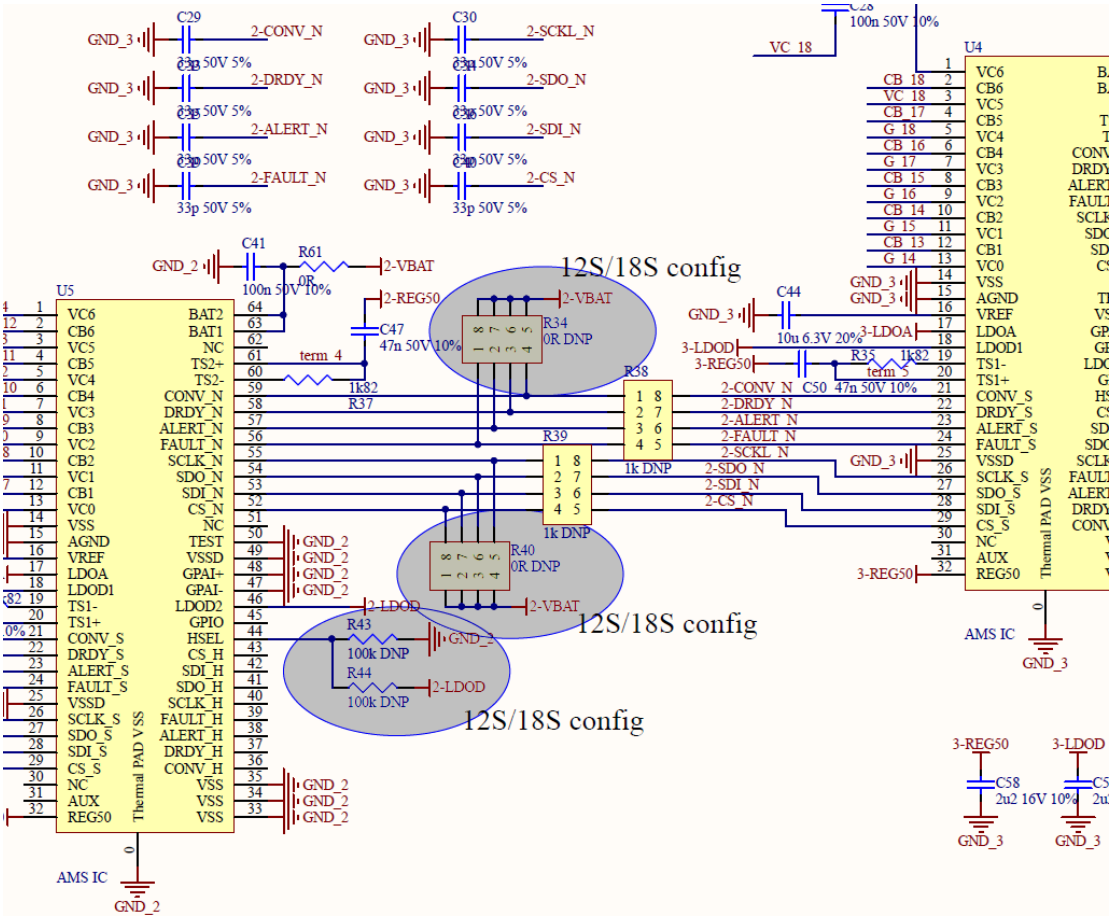
Probléma volt viszont, hogy a státusz-LED-ek (OT – Over Temperature, OV – Over Voltage, UV – Under Voltage) nem megfelelően működtek. Itt a processzor nem megfelelő lábkiosztásának kijavítása megszüntette a hibát. Ez esetben a működést úgy teszteltem, hogy a kisfeszültségű oldalt tápláló tápegység feszültségértékét letekertem 9.6V alá és megnéztem, felvillan-e ekkor az UV (Under Voltage) LED. Ennek a LED-nek akkor is jeleznie kell, ha a cellafeszültségek közül valamelyik a config.h fájlban beállított `PARAMETER_CELL_UNDERVOLTAGE_THRESHOLD` makró értéke alá esik. Ezt pedig úgy teszteltem, hogy az akkumulátor-cellákat szimuláló áramkört tápláló tápegység kimeneti feszültségét csökkentettem a beállított érték alá. Hasonlóan teszteltem az OV (Over Voltage) LED működését is. Az OT (Over Temperature) LED működésének ellenőrzéséhez pedig a hőmérő ellenállást beleraktam egy pohár forró vízbe és figyeltem, hogy felvillan-e a LED.

A következő lépésben az áramkör kimeneteit teszteltem, melyek a következők: CH_EN, DR_EN, DR_EN_DELAYED. Ezek teszteléséhez szintén az akkumulátort szimuláló áramkört meghajtó tápegység feszültségszintjét változtattam. A vártnak megfelelő volt a működés, vagyis a `PARAMETER_CELL_STOP_CHARGING_THRESHOLD` makróval beállított 4,1V feletti cellafeszültség esetén a CN_EN kimenet lekapcsolt, és a `PARAMETER_CELL_UNDERVOLTAGE_THRESHOLD` makróval beállított 3,65V alatt pedig a DR_EN kimenet kapcsol le.

A kimenetek tesztelését követően a Kvaser Canking nevű program segítségével leellenőriztem a CAN periféria működését. Ekkor kiderült, hogy a 640-es azonosítójú CAN üzenet is megjelenik, amely csak inicializációs hiba esetén kéne, hogy kiküldésre kerüljön, ez viszont az jelen esetben nem volt. A hiba javításának érdekében a freertos.c

fájlban a taszkok létrehozásánál beállítottam, hogy az említett taszk csak inicializációs hiba esetén jöjjön létre. A CAN periféria ezt követően hiba nélkül működött.

A következő lépésben az akkumulátort szimuláló áramkört kiegészítettem még 6db ellenállással, hogy teszteljem a 18 cellás funkciót. 18 darab cella használata esetén a szoftvernek már 3 darab BMS IC-re van szüksége. Ehhez egyrészt át kell állítani a szoftverben a `PARAMETER_CELL_COUNTS_2` makrót 0-ról 6-ra (így jelezzük a programnak, hogy a 3. cella-csomag 0 db. cella helyett 6-ot tartalmaz), illetve a `PARAMETER_TEMP_CHECK_2` makrót `TEMP_CHECK_BOTH_OT_DISABLED`-ról `TEMP_CHECK_BOTH_OT_ENABLED`-re (annak érdekében, hogy a harmadik 6-os cellacsomag hőmérőit aktiváljam), másrészt az áramkörön két ellenállás-létrával össze kell kapcsolni az U5-ös és az U4-es BMS IC-t



5.1 ábra: A BMS IC-k közötti összeköttetést módosítva lehet megválasztani a használni kívánt akkumulátor-cellák számát [10]

Ehhez az 5.1-es ábrán látható R34-es és R40-es ellenállást ki- az R38-as és R39-es ellenállást pedig be kellett forrasztani. Ezt követően a két BMS IC közötti SPI vonal nem lesz leföldelve, így képesek lesznek egymással kommunikálni. A kapcsolási rajzon a

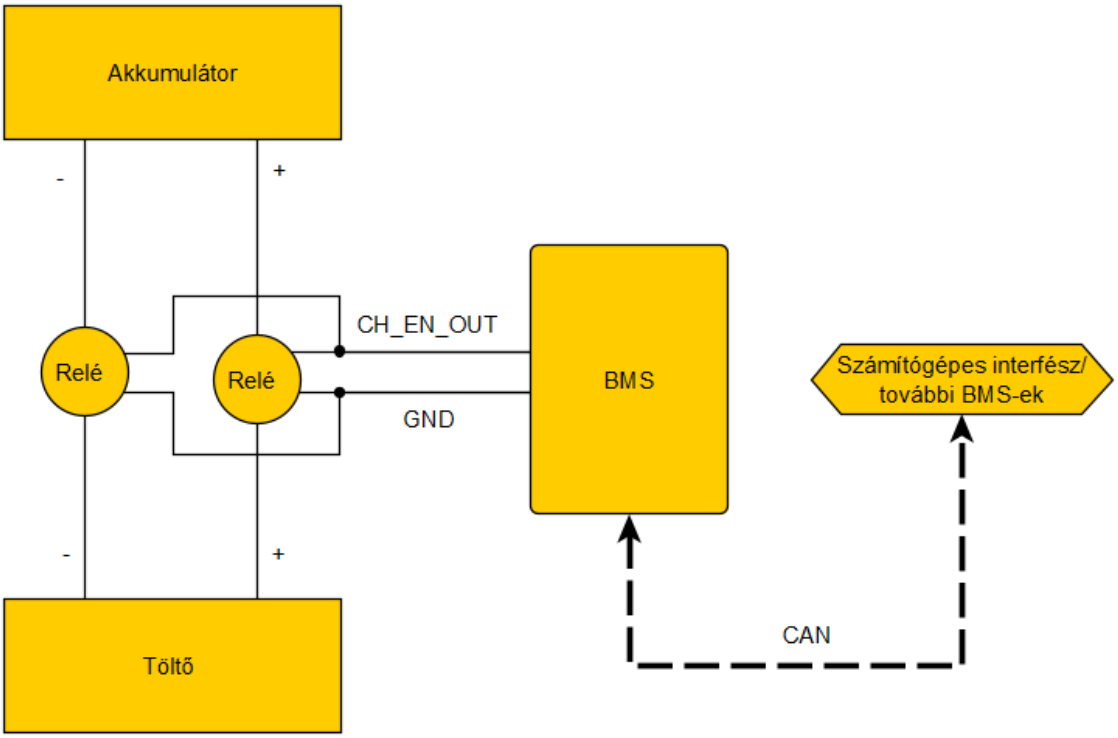
szürke karikázások jelzik, hogy 12-, illetve 18-as cellakonfiguráció esetén mely áramköri elemeket kell átforrasztani. A tesztelés folyamán, viszont szembesültem azzal, hogy az 5.1.-es ábrán látható bal oldali IC HSEL lábát 12, illetve 18 cellás beállítás mellett is tápra kell kötni, tehát az R43-mas ellenállás fölösleges, így a karikázás sem lenne szükséges.

Miután a fent említett beállításokat kiviteleztem, a korábban bemutatott tesztelési módszerrel leellenőriztem a BMS működését.

Mivel a működés helyesnek bizonyult, ezért a következő lépésben az akkumulátor-csomagot szimuláló áramkört kicseréltem egy valódi akkumulátor-csomaggal, amelyet az Xtalin Kft. biztosított és a következő tulajdonságokkal rendelkezett:

- 18db Li-Po cella sorbakötve
- Valamennyi cella névleges feszültsége: 3,7V
- kapacitás cellánkként: 1100mAh
- maximális áram: 1,1A
- maximális feszültség: 4,2V
- minimális feszültség: 2,9V
- minden akkumulátor-cellán rendelkezésre áll PCB védelem, amely 2,75V-os minimális- és 4,3V-os maximális töltöttséget engedélyez
- Névleges töltési és merítési áram: 0,5A

Az akkumulátor-cellák tulajdonságaiból következik, hogy a csomag maximális feszültsége 4,2V*18=75,6V, ami azt jelenti, hogy a CH_EN és a DR_EN jelet kapcsoló reléknek minimum ekkora feszültséget kell tudniuk kapcsolni. A 0,5A-es töltési, illetve merítési áramból pedig következik, hogy a relének el kell tudnia viselni legalább ekkora áramot. Az alkalmazott relé 100V-ig működik, 10A nagyságú áram kapcsolására képes és 12V-tal kapcsolható, így bőven megfelel a fenti követelményeknek. Mind a töltést és a terhelést kapcsoló relék az akkumulátor pozitív pólusát szakítják meg.



5.2. ábra: A töltést vezérlő relé és a BMS viszonya (a valóságban a negatív pólust is meg kell szakítani)

Ahhoz, hogy tesztelni tudjam az akkumulátor-csomagnak a töltését, a config.h fájlban be kell állítani a fent említett értékekkel összhangban lévő makrókat. Ezeknek a következő értékeket választottam (az értékek millivoltban vannak megadva):

```
PARAMETER_CELL_OVERVOLTAGE_THRESHOLD:                    4200
```

Ez az abszolút maximum feszültség, ami engedélyezhető egy cellának, e felett az érték felett az OV (Over Voltage) LED felvillan.

```
PARAMETER_CELL_UNDERVOLTAGE_ALERT_THRESHOLD                    3000
```

Ez alatt a feszültség alatt az underVoltage flag aktiválódik.

```
PARAMETER_CELL_UNDERVOLTAGE_THRESHOLD              2900
```

Ez az abszolút minimum feszültség, ami engedélyezhető egy cellának, ez alatt az UV (Under Voltage) flag aktiválódik és a DR_EN kimenet lekapcsol.

```
PARAMETER_CELL_STOP_CHARGING_THRESHOLD             4100
```

E felett a feszültség felett a töltés kikapcsol.

```
PARAMETER_CELL_START_CHARGING_THRESHOLD            4000
```

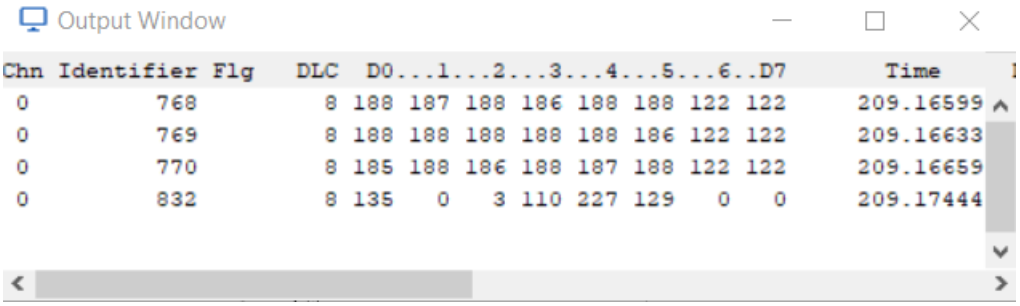
A töltés elindításához, minden cellának a feszültsége ez alatt kell legyen. Abban az esetben, ha a töltés már korábban elindult, az értéknek nincs hatása.

```
PARAMETER_CELL_CHARGING_UNDERVOLTAGE_THRESHOLD          2750
```

Ez alatt a feszültségszint alatt a töltés tiltásra kerül.

A sikeres töltéshez aktiválni kell a programban a kiegyenlítést a `DISABLE_BALANCING` makró kikommentelésével.

Miután a szoftvert maradéktalanul felkészítettem a tesztre és a relét is beiktattam a tápegység és az akkumulátor közé, a tápegység két csatornáját sorba kötöttem (mivel a használt tápegység egy csatornája maximum 60V-ig működik) és beállítottam 0.5A-es áramkorlátot, hogy a töltés folyamatosan 0.5A-el történjen. A BMS-ből érkező adatokat pedig CAN-en keresztül monitoroztam a Kvaser Canking nevű program segítségével, így láthattam, hogy az egyes cellák milyen töltöttségi szinten vannak, illetve, hogy a korábban beállított feszültségértékek elérése esetén jól reagál-e az áramkör.



5.3. ábra: A CAN perifférián érkező üzenetek

Az 5.3. ábrán látható, hogy a 768, 769 és 770-es (hexadecimális formátumban 300, 301 és 302-es) azonosítóval rendelkező üzenetek első 6 bájtja 6 darab cella feszültségét tartalmazza, utolsó két bájtja pedig az egy-egy hőmérő által mért hőmérsékletet. A 832-es (hexadecimális formátumban 340-es) üzenet pedig a státusz-üzenet. Egy cella feszültségét Volt-ban úgy kapjuk meg, hogy a CAN-en érkező adathoz hozzá kell adni 200-at, majd osztani kell 100-zal. Például a 768-as azonosítóval rendelkező CAN-üzenet első bájtja (D0) hordozza az első cella feszültségét, ami ebben az esetben (188+200) /100=3,88V

Ahhoz viszont, hogy tesztelni tudjuk a BMS működését, meg kell értenünk, hogyan működik pontosan a kiegyenlítés, melyet a következő pontban ismertetek.

5.1 A cellakiegyenlítés menete

Az akkumulátor-cellák kiegyenlítésének az a célja, hogy valamennyi cella azonos feszültségszinten legyen, amelynek az az előnye, hogy a terhelés folyamán – kiegyenlítés

mellett – nem fordulhat elő az a helyzet, hogy van egy cella amelyik már teljesen lemerült, így az tovább már nem is terhelhető, a többi cella viszont még rendelkezik jelentős töltéssel. Ez pedig azért kellemetlen, mert ha az akkumulátorcellák sorba vannak kötve, az egy lemerült cella miatt az egész akkumulátor nem terhelhető, hiába van több olyan cella is, amelyek még töltöttek. Ezt a helyzetet hivatott megoldani a BMS áramkör, amely ezáltal az akkumulátor-csomag üzemidejét növeli.

Egy cellának a kiegyenlítése annyit jelent, hogy – bizonyos feltételek teljesülése esetén – az adott cellát egy ellenálláson keresztül kisütjük, addig amíg újra egyik feltétel sem teljesül. A kiegyenlítés tényét – esetünkben – a BMS 18 darab LED-el képes jelezni a felhasználó számára, ami a tesztelés során sokat fog segíteni.

Most pedig nézzük, milyen feltételeknek kell teljesülnie, ahhoz, hogy egy cellát kiegyenlíteni kelljen.

- Először is az adott cella feszültsége nagyobb kell legyen a `PARAMETER_BALANCE_CELL_VOLTAGE_MARGIN` makró értékénél, ami esetünkben 3,9V-ra van állítva. Ez lényegében annyit jelent, hogy amíg egy cella el nem éri ezt a feszültségszintet, addig kiegyenlítés nélkül zajlik a töltés.
- A második feltétel, aminek teljesülnie kell a kiegyenlítéshez, pedig valójában három feltétel, melyek vagy-kapcsolatban állnak egymással. Ezek pedig a következők:
    1. Az adott cella és a legkisebb feszültségű cella közötti feszültség-különbség nagyobb kell legyen a `PARAMETER_BALANCE_CELL_VOLTAGE_MARGIN_MIN_DIFFERENCE` makró értékénél, amely esetünkben 20mV-ra van állítva.
    2. Az adott cella feszültsége nagyobb, mint a `PARAMETER_CELL_STOP_CHARGING_THRESHOLD` makró értéke, ami esetünkben 4,1V.
    3. Kényszerített kisütés flag aktív (ez akkor fordulhat elő, ha több BMS-t sorba kötve használunk, tehát esetünkben ez a feltétel sosem teljesül).

Miután megismertük a kiegyenlítés működési elvét, képesek leszünk megmagyarázni a tesztelés során tapasztaltakat. A töltés megkezdése előtt minden cella feszültsége nagyjából 3,7V. A töltés megkezdésekor a cellafeszültségek felugranak 3,85V-ra a cellák

belső ellenállása miatt. Cella-kiegyenlítés mindaddig nem történik, amíg a cellák feszültsége el nem éri a 3,9V-os feszültséget, így ez alatt az idő alatt egyik cella balansz-LED-je sem villan fel. Amikor a tesztelés során az akkumulátor-cellák feszültsége túllépi a 3,9V-os szintet még mindig nem történik kiegyenlítés, de ha jobban megnézzük (a CAN csatornán érkező adatok alapján) a cellák közti feszültség-különbség kisebb, mint a szoftverben beállított 20mV, e-miatt, hogy teszteljem a működést ezt az értéket levittem 10-re. Ezt követően a kiegyenlítés gyakori volt, minden cellán. Amikor az egyik cella feszültsége elérte a 4,1V-ot, a CH_EN kimenet lekapcsolt, az általa vezérelt relé pedig megszakította a töltést. Ekkor a cellák feszültsége újra visszaesett 3,9 körüli értékre, mivel a töltőáram megszűnt, ami pedig a belső ellenállásukon átfolyva, feszültséget ejtett. Ugyanakkor a `PARAMETER_CELL_START_CHARGING_THRESHOLD` makrónak 4000-es érték volt beállítva, aminek köszönhetően a töltés csak 4V feletti érték esetén nem indul el, így amint megszakadt a töltés és visszaesett a cellák feszültsége 3,9-re, már vissza is kapcsolta a szoftver. Ez a relé folyamatos ki-be kapcsolását eredményezi, melynek elkerülése érdekében a 4 Voltos küszöbértéket 3,9-re állítottam. Ezt követően megfelelő volt a működés, így a tesztelést sikeresen fejeztem be.

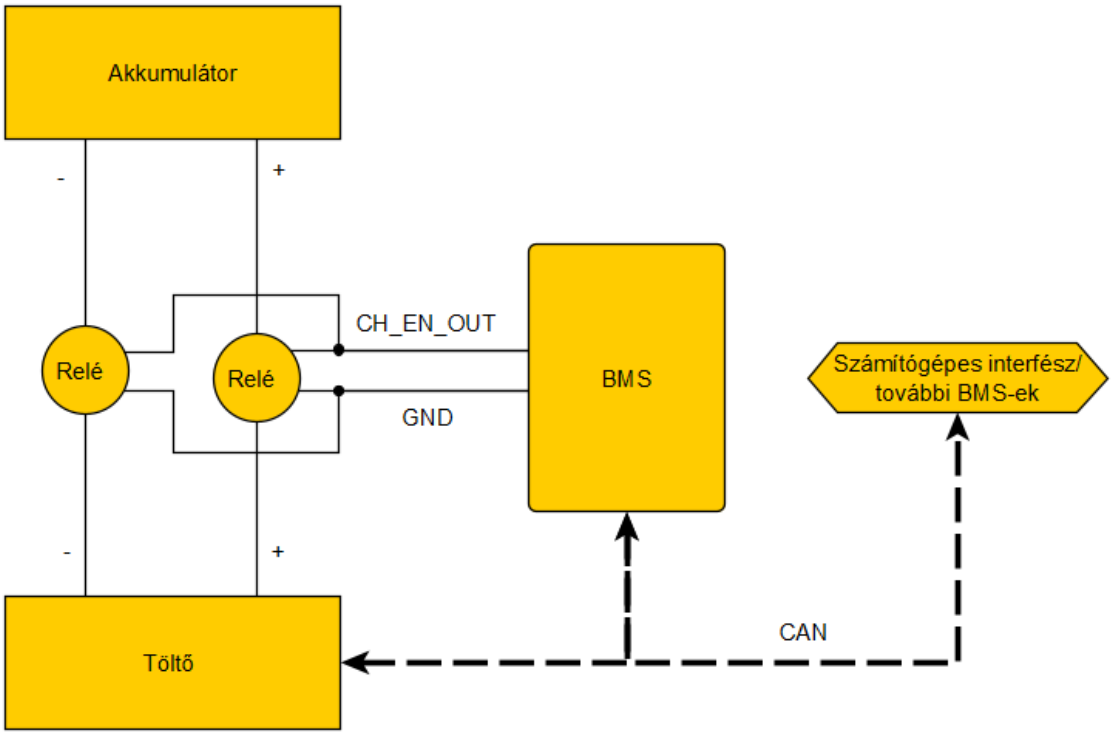
6 Fejlesztési lehetőségek

6.1 Töltés

A jelenlegi konfiguráció mellett a töltés addig tart, amíg valamelyik cella feszültsége el nem éri a 4.2V-os értéket. Ezt követően a töltés azonnal lekapcsol és az akkumulátor-cellák belső ellenállása miatt a feszültségük – a töltőáram nagyságától függően – visszaesik. A teszt során a cellák feszültsége ekkor 3.95V körüli értékre estek vissza, ami azért probléma, mert a cellák maximális töltöttségüket 4.2V-os feszültség-szintnél érik el, így a töltés jóval 100% alatt fejeződik be.

Erre megoldás lehet, ha ekkor a töltőáramot csökkentjük, így a cellák belső ellenállásán kisebb feszültség esik, emiatt pedig magasabb töltöttségi szintet tudunk elérni. Ehhez persze az kell, hogy kommunikálni tudjunk a töltővel, amihez további kimenetek szükségesek.

Tekintve, hogy az autóiparban elterjedt kommunikációs protokollnak számít a CAN és mindemellett viszonylag hosszú vezetékezés mellett is használható (tudva, hogy a töltővezeték pár métertől akár 10 méterig is terjedhet a legtöbb esetben), célszerű a használata. Továbbá mivel a CAN egy olyan kommunikációs protokoll, aminél nem a buszon jelenlévő eszközök vannak azonosítva, hanem az egyes üzenetek, így valamennyi üzenetet minden a buszon jelenlévő eszköz megkap, amiből az is következik, hogy új eszközt bármikor könnyedén lehet csatlakoztatni.



6.1 ábra: A töltő vezérlése CAN-busz segítségével

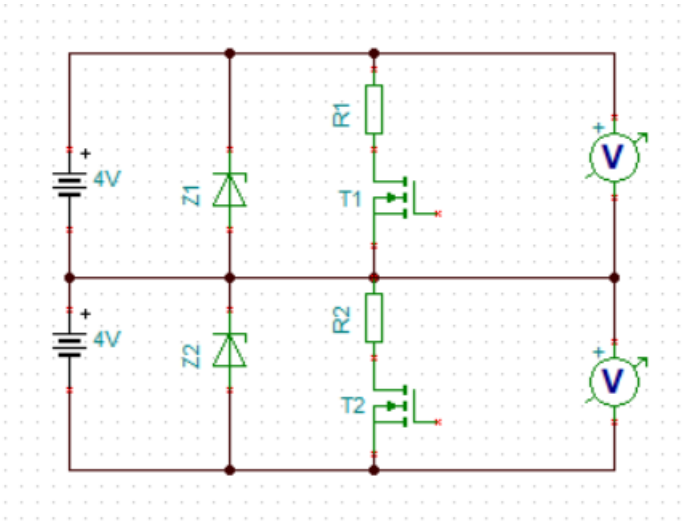
Emiatt pedig ez a funkció hardveres változtatások nélkül is megvalósítható, tehát csak a szoftvert kell módosítani. Ez pedig azért előnyös, mert a NYÁK módosítása egy pénzügyileg drága művelet a gyártásból adódóan.

Ehhez persze feltétel az, hogy rendelkezésre áll egy olyan töltőegység, amellyel CAN buszon keresztül lehet kommunikálni. Ha egy ilyen töltő adott a számunkra, az ebben a pontban részletezett fejlesztési lehetőség értelmet nyer.

6.2 Cellák összeköttetésének ellenőrzése

Szintén hasznos funkció lehetne, ha detektálni tudnánk azt, hogy egy cella és a BMS közötti összeköttetés megszakadt. Ebben az esetben hibás mérési adatok fognak megjelenni annak a két cella feszültségére, melyek közötti mérő-vezeték megszakadt. A szakadás megállapításához felhasználhatók a balansz-ellenállások, amihez a hozzájuk tartozó FET tranzisztorokat kell kapcsolgatni. Fontos, hogy a mérővezetékek összeköttetését még a cellakiegyenlítés előtt leellenőrizzük, hiszen a kiegyenlítés során, szakadás esetén az adott Zener-dióda túlterhelődhet, mivel az csak egy rövid ideig terhelhető. Ebből az is következik, hogy az összeköttetések vizsgálatát kellően gyorsan kell elvégezni, amely a következőképpen fog történik:

Mindig egyszerre csak az egyik tranzisztort kapcsoljuk be, ezt követően pedig megmérjük a hozzátartozó cellafeszültséget. A 6.2. ábrát tekintve tegyük fel, hogy a középső mérővezeték van elszakadva és a felső tranzisztort bekapcsoltuk. Ekkor az áram útja a Z2-es diódán keresztül záródik, így az alsó cellára 6V-ot a felső cellára pedig 2V körüli értéket mérünk, tudva, hogy a Zener-diódák 6V-osak. Ha az alsó tranzisztort kapcsoljuk be, akkor a felső cellán 6, az alsón pedig 2 V-ot mérünk. Ilyen mérési eredmények esetében tudjuk, hogy a középső mérővezeték van elszakadva. Most tegyük fel, hogy az alsó mérővezeték van elszakadva. Ekkor az alsó cellát tekinthetjük úgy mintha nem is lenne az áramkör része, így alul 0V-ot mérünk, a felső cellát viszont 4V-ra mérjük. A felső vezeték szakadása esetén pedig a felső cellán 0 az alsón pedig 4V-ot mérünk mind a felső és az alsó tranzisztor bekapcsolása esetén.



6.2. ábra: Két cella feszültségének méréséért és kiegyenlítéséért felelős áramkör

Mindebből következik, hogyha két egymás melletti cellákat 6 és 2V-ra mérünk, akkor a két cella közötti vezeték szakadt el. A legfelső és a legalsó vezeték szakadását pedig úgy állapíthatjuk meg, ha a legfelső, illetve legalsó cellákra 0V-ot mérünk.

A BMS IC-knek elküldött SPI üzenettől számolva, a mérési eredmények megérkezéséig, majd az újabb SPI üzenet visszaküldéséig eltelt idő alatt egy-egy Zener-dióda terhelődhet szakadt mérővezeték esetén. Emiatt a mérési eredmények feldolgozását és a megfelelő beavatkozást előidéző üzenet kiküldését minél gyorsabban kell elvégezni.

7 Konklúzió, köszönetnyilvánítás

Ezen szakdolgozatot annál az Xtalin Kft-nél írtam, amelynél a nyári szakmai gyakorlatomat is teljesítettem. A szakmai gyakorlatom leteltét követően a cég megtisztelt az ezen szakdolgozat tárgyát képező projekt felajánlásával, melyet örömmel fogadtam.

A BMS áramkörének módosítása, illetve az azt követő szoftver átírása sikeres volt melyet a tesztelés is igazolt. A működést 12, illetve 18 cellás akkumulátor-csomag esetében is teszteltem. A töltés minden alkalommal addig tartott, amíg az egyik cella feszültsége el nem érte a config.h fájlban definiált maximális cellafeszültséget. Ezt követően a cellák kiegyenlítése folytatódott, a töltés pedig akkor indult újra, amikor minden cella feszültsége a töltés újraindításához szükséges maximális feszültség alá esett. Megfelelően működtek továbbá a cellák, illetve a táp túl kis- vagy nagy feszültségszintjét jelző LED-ek, valamint az akkumulátor terhelése esetén a terhelést engedélyező kimenetek. Az optimális működés az áramköri átalakítások helyességét is bizonyítják.

A szakdolgozat írása során megismerkedtem többek között az STM32CubeMX, az Atollic trueSTUDIO és az ATMEL Studio nevű programokkal és fejlesztőkörnyezetekkel. Használtam továbbá a tesztelés alkalmával Picoscope-ot, illetve a Kvaser Canking programot a CAN perifférián érkező üzenetek olvasására. Megismertem a BMS működési elveit és a korábban használt ATMEL, illetve a jelenleg is használt STM mikroprocesszor perifériáit és a rendelkezésre álló függvénykönyvtárakat.

Ezen projekt sikeres kivitelezése az Xtalin Kft. támogatása nélkül elképzelhetetlen lett volna, melyet ezúton is köszönök. Köszönöm továbbá az Xtalin Kft munkatársainak és persze a külsős konzulensemnek, Szabó Ádámnak, hogy a szakdolgozatom írása során felmerülő kérdéseimben minden alkalommal fáradhatatlanul segítettek, továbbá, hogy a munka elvégzéséhez megfelelő környezetet biztosítottak számomra. Végül köszönöm a segítséget az egyetemi konzulensem, Dr. Renczes Balázs részéről, a szakdolgozatom számtalan átolvasásáért és az iránymutatásokért.

Irodalomjegyzék

[1] Xtalin Kft.: BMS dokumentáció (nem publikus), 13. oldal

[2] ATMEL: 8-bit Microcontroller with 128K Bytes of ISP Flash and CAN Controller, Rev. 7522C–AUTO–09/06, http://biakom.com/pdf/AT90CAN128-15AZ_Atmel.pdf

[3] ST: DocID022152 Rev 8, https://www.st.com/resource/en/datasheet/dm00037051.pdf

[4] Az ATMEL mikroprocesszor lábkiosztása az Xtalin Kft.: BMS kapcsolási rajz és bekötés (nem publikus) dokumentumának 7. oldalán található

[5] ATMEL: 8-bit Microcontroller with 128K Bytes of ISP Flash and CAN Controller, Rev. 7522C–AUTO–09/06, http://biakom.com/pdf/AT90CAN128-15AZ_Atmel.pdf

[6] ST: RM0090 Reference manual, https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

[7] ST: DocID022152 Rev 8, https://www.st.com/resource/en/datasheet/dm00037051.pdf

[8] en.STM32F7_Peripheral_bxCAN: 7. oldal, https://www.st.com/content/ccc/resource/training/technical/product_training/group0/49/99/d2/9b/67/7a/48/c0/STM32F7_Peripheral_bxCAN/files/STM32F7_Peripheral_bxCAN.pdf/jcr:content/translations/en.STM32F7_Peripheral_bxCAN.pdf

[9] ST: RM0090 Reference manual, https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

[10] Xtalin Kft.: X_BMS_18S_v5_schematics.pdf (nem publikus), 42. oldal

[11] Xtalin Kft.: X_BMS_18S_v5_schematics.pdf (nem publikus), 41. oldal