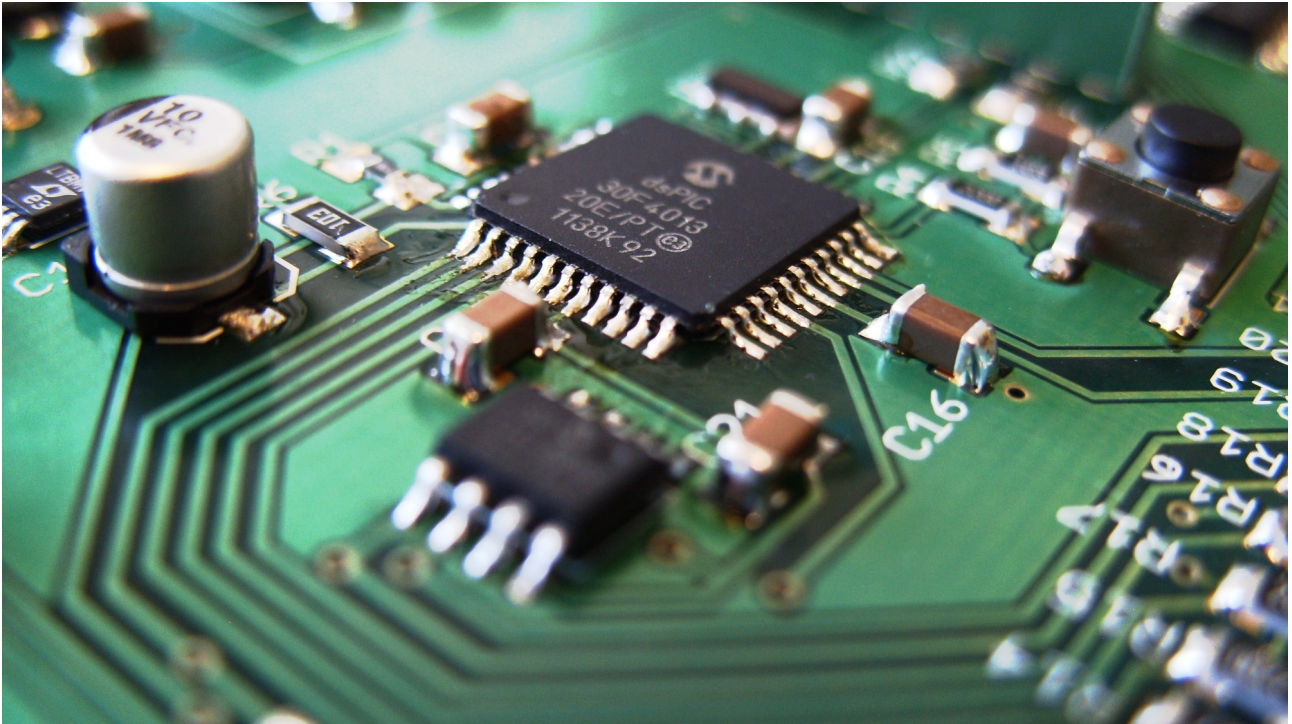


CHALMERS



Battery Status Module

Utveckling av ett batteriövervakningssystem med CAN-kommunikation

Examensarbete inom högskoleingenjörsprogrammet Elektroingenjör

Anders Reinholdsson

Johan Persson

Institutionen för signaler och system
Avdelningen för digitala bildsystem och bildanalys
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2012

Förord

Denna rapport beskriver genomförandet av ett examensarbete som utfördes vid Xdin under våren 2012. Examensarbetet utgör det avslutande momentet i Anders Reinholdssons och Johan Perssons elektroingenjörsexamen vid Chalmers tekniska högskola. Arbetet omfattar 15 högskolepoäng.

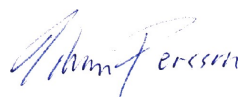
Vi vill tacka vår handledare Jacob Hägglund vid Xdin för handledning och stöd genom hela projektet. Ett speciellt tack ges för den ambitiösa handledningen under kretskortsutvecklingen, ett moment i genomförandet vi inte hade någon tidigare erfarenhet av.

Ett tack går även till vår handledare Manne Stenberg vid Institutionen för signaler och system, Chalmers tekniska högskola för goda synpunkter under mittmötet och vägledning vid utformandet av examensrapporten.

Övriga tack går till Jonas Johansson för sin positiva inställning till projektet, Fredrik Lönnblad för information om radiobilens CAN-system, Fredrik Engström för medverkan vid sammansättning och test av batteriövervakningssystemet på radiobilen och Springer Science för tillåtelse att använda figurer.



.....
Anders Reinholdsson



.....
Johan Persson

Sammanfattning

Ökande miljökrav på bilindustrin har intensifierat utvecklingen av tillförlitliga elbilar och elhybrider. För att batteridrivna fordon ska kunna nyttjas på ett effektivt sätt krävs kontinuerlig rapportering av bland annat återstående körtid och batteriets kondition. Syftet med examensarbetet är att utveckla ett system för batteriövervakning som tillhandahåller denna information till användaren. Arbetet är utfört vid Xdin som är ett konsultföretag med verksamhet inom fordonsindustrin. Projektet inriktar sig inte på att utveckla det perfekta systemet för batteriövervakning, utan är mer inriktat på den teori och konstruktionsprocess som utvecklingen av ett sådant kräver. Projektets frågeställning är huruvida det är möjligt, att med 10 veckors arbete, utveckla en modul som i stora drag kan uppskatta laddningsgrad, återstående körtid och hälsostatus för ett batteri i en elbil. Hårdvaran i systemet är framtagen från grunden med en mikrokontroller som kärna i systemet. Fokus har legat på noggrann ström- och spänningsmätning. Mjukvaran är utvecklad med omfattande litteraturstudier som grund. Implementationen är gjord i programspråket C. Resultatet av arbetet är en modul för batteriövervakning med kommunikation via CAN och RS-232. Modulen loggar kör-data till ett minne och är konfigurerbar med hjälp av ett menysystem. Tester av systemet visar på mycket god noggrannhet vad gäller ström- och spänningsmätning. Begränsade tester ger en indikation på god noggrannhet i uppskattningen av batteriets kondition.

Abstract

Increasing environmental demand on the car industry has intensified the need for dependable electric and hybrid vehicles. In order to make sure that battery powered vehicles are used in the most effective way, careful monitoring and forwarding of data such as remaining run time and battery health are necessary. The purpose of the degree project is to develop a module that is capable of serving this information to the end user. The project is executed at Xdin which is a consulting company with operation in the vehicle sector. The objective of the project is not to develop the perfect battery management system, but more to gain insight in the theory and construction process needed in the development of such a system. The main question of the project is in what extent it is possible, in ten weeks time, to develop a module that approximately can estimate the state-of-charge, remaining run time and the state-of-health of a battery in an electric vehicle. The system hardware is based around a microcontroller. Furthermore the hardware is designed with accuracy of current and voltage measurements in mind. The software development is made with a foundation of in depth literature studies and implemented in the programming language C. The result of the project is a battery management system with CAN and RS-232 communication. The module stores real time driving information to a memory and is adjustable through a menu system. Test results prove good accuracy in the current and voltage measurements. Restricted tests give an indication of a good accuracy in the prediction of the state of charge of the battery.

Innehållsförteckning

Beteckningar.....	1
1 Inledning.....	3
1.1 Bakgrund.....	3
1.2 Syfte.....	3
1.3 Avgränsningar.....	4
1.4 Mål.....	4
1.4.1 Grundläggande mål.....	4
1.4.2 Högre mål.....	5
2 Teknisk bakgrund.....	6
2.1 Algoritmer och metoder.....	6
2.1.1 Direkta mätningar.....	7
2.1.2 Bokförande system.....	9
2.1.3 Adaptiva system.....	9
2.2 Fixpunktsaritmetik.....	10
2.3 Litiumjonbatteriet.....	11
2.3.1 Bakgrund.....	11
2.3.2 Uppbyggnad.....	12
2.3.3 Icke-ideala faktorer.....	12
2.3.4 Funktion.....	13
2.3.5 Laddning av litiumjonbatterier.....	14
2.3.6 Krav på säkerhet.....	15
2.4 Spännings- och strömmättningsmetoder.....	16
2.5 Sampling, samplingsteoremet och antialiasningsfilter.....	18
2.6 A/D-omvandling.....	20
2.7 Icke ideala faktorer hos operationsförstärkare.....	21
2.8 Kommunikation.....	22
2.8.1 CAN.....	22
2.8.2 RS-232.....	23
3 Val av komponenter och utvecklingsmiljö.....	25
3.1 Komponenter.....	25
3.1.1 Mikrokontroller.....	25
3.1.2 Ström och spänningsmätning.....	25
3.1.3 A/D-omvandling.....	26
3.1.4 Flashminne.....	26
3.1.5 Övriga komponenter.....	27
3.2 Utvecklingsmiljö.....	28
3.2.1 MPLAB IDE.....	28
3.2.2 Programmerare.....	28
4 Framtagning av elektriskt schema.....	29
4.1 Mikrokontroller, Microchip dsPIC30F4013.....	29
4.2 Spänningsregulator, Micrel MIC29150.....	30
4.3 Spänningsdelning.....	30
4.4 Strömförstärkare, Linear Technology LT6100.....	32
4.5 Reset-nät.....	33
4.6 Nivåomvandlare för RS-232-kommunikation, Texas Instruments MAX232.....	34
4.7 Nivåomvandlare för CAN-kommunikation, Microchip MCP2551.....	35
4.8 Flashminne, Microchip 24LC128.....	36

4.9 Headers.....	36
5 Konfiguration av elektriska kretsar.....	38
5.1 Mikrokontroller, Microchip dsPIC30F4013.....	38
5.1.1 Konfiguration av klockfrekvensen.....	38
5.1.2 Konfiguration av portar.....	38
5.1.3 Konfiguration av UART-modulen.....	39
5.1.4 Konfiguration av analog till digital-omvandlaren.....	39
5.1.5 Konfiguration av I2C-modulen.....	40
5.1.6 Konfiguration av räknare.....	41
5.1.7 Konfiguration av CAN-modulen.....	43
5.1.8 Konfiguration av avbrott.....	44
5.2 Nivåomvandlare för RS-232-kommunikation, Texas Instruments MAX232.....	45
5.3 Strömförstärkare, Linear Technology LT6100.....	45
5.4 Spänningsregulator, Micrel MIC29150.....	45
5.5 Flashminne, Microchip 24LC128.....	45
5.6 Nivåomvandlare för CAN-kommunikation, Microchip MCP2551.....	45
6 Prototyputveckling.....	46
7 Kretskortskonstruktion.....	48
7.1 Val av komponenter.....	48
7.2 EAGLE.....	48
7.3 Konstruktionsförfarande.....	49
7.4 Resultat av kretskortskonstruktion.....	50
8 Algoritmutveckling.....	53
8.1 Starttillstånd.....	54
8.2 Transitionstillstånd.....	54
8.3 Vilotillstånd.....	54
8.4 Urladdningstillstånd.....	55
9 Kommunikation.....	56
9.1 CAN.....	56
9.2 RS-232.....	58
10 Signalbehandling.....	59
11 Loggning till flashminne.....	62
11.1 Funktioner för läsning och skrivning av en byte.....	62
11.2 Funktioner för läsning och skrivning av en post.....	62
11.3 Funktioner för reset och formatering.....	63
11.4 Funktioner för lagring och utskrift av kördata.....	63
11.5 Data som lagras på flashminne.....	64
12 Testning.....	65
12.1 Kalibrering och test av spänningsmätning.....	65
12.2 Kalibrering och test av strömmätning.....	67
12.3 Uppmätning av driftström.....	69
12.4 Test av algoritmen.....	69
13 Specifikationer.....	72
14 Slutsats och diskussion.....	73
Referenser.....	74
Bilagor.....	77
Elektriskt schema.....	77

Källkod.....	78
main.c.....	78
macros.h.....	99
can.h.....	101
can.c.....	102
eeprom.h.....	103
eeprom.c.....	105
io.h.....	110
io.c.....	111
Användningsinformation.....	116
Tidplan.....	117

BETECKNINGAR

C	Kapacitet	[Ah]
CAN	Controller Area Network	
CCCV	Constant-Current-Constant-Voltage	
CM	Common-mode	[V]
CMRR	Common-mode Rejection Ratio	[dB]
DM	Differential-mode	[V]
dsPIC	Digital Signal Processing Peripheral Interface Controller	
EEPROM	Electrically Erasable Programmable Read-Only Memory	
EID	Extended Identifier	
EMF	Electro-Motive Force	[V]
GND	Ground Signaljord	
I₂C	Inter-Integrated Circuit	
I_{AVG}	Medelström under körning	[A]
ICSP	In-circuit serial programming	
IDE	Integrated Development Environment	
LiCoO₂	Lithium Cobalt Oxide	
OSC	Oscillator	
PCB	Printed Circuit Board	
PIC	Peripheral Interface Controller	
PTC	Positive Temperature Coefficient	
Q_{AVA}	Tillgänglig kapacitet	[Coulomb]
Q_{NOM}	Nominell kapacitet	[Coulomb]

Q_{OUT}	Genomfluten laddning	[Coulomb]
Q_{TOT}	Verklig kapacitet	[Coulomb]
RX	Receive Port	
SID	Standard Identifier	
SoC	State-of-Charge Laddningsgrad	[%]
SoH	State-of-Health Hälsostatus	[%]
t_r	Återstående körtid	[s]
TX	Transmit Port	
UART	Universal Asynchronous Receiver/Transmitter	
V_{DD}	Voltage Drain-Drain Positiv matningsspänning	

1 INLEDNING

1.1 Bakgrund

Det finns ett ökande behov av kompetens inom övervakning av batterier. Mycket av detta beror på miljökraven som ställs på bilindustrin, där hybridbilar och elbilar blir vanligare. Vid användande av batterier i bilar är det viktigt att kunna uppskatta kvarvarande körsträcka och bedöma batteriets skick.

Xdin är ett konsultföretag som jobbar med produktutveckling, analys, simulering, elektronik och mjukvaruutveckling. Företaget har ett behov av ökad kompetens inom området batteriövervakning. I ett kompetensutvecklingsprojekt utvecklas fristående moduler som kommunicerar med varandra via CAN. Utvecklingen av modulerna sker mot en elektrisk radiobil för att efterlikna den elektriska miljö som finns i en fullstor elbil. I detta projekt finns liksom i en modern hybrid/elbil ett behov av batteriövervakning. Detta examensarbete går ut på att utveckla ett sådant batteriövervakningssystem för Xdin.

1.2 Syfte

Syftet med projektet är att konstruera en modul för batteriövervakning som ska fungera i ett större system. Med hjälp av CAN-kommunikation ska modulen kunna rapportera diverse information om systemets batteristatus. Projektet är tänkt att ha en flexibel omfattning och anpassas efter hur arbetet utvecklas. I grundutförandet ska systemet kunna ge en ungefärlig uppskattning av aktuell laddningsgrad och kommunicera via en CAN-buss. Om de grundläggande kraven uppfylls är idén att projektet ska kunna utvidgas till att även innefatta några av de högre målen. I delen avgränsningar preciseras vad som avses med de grundläggande respektive högre målen.

Från XDIN:s sida är det främsta syftet med projektet att utveckla en hårdvara till en modul som skall kunna mäta spänning och ström noggrant samt kommunicera via CAN. Modulen kommer då att möjliggöra en vidare utveckling med avseende på algoritm och mjukvara.

Vårt syfte med projektet är att konstruera denna hårdvara samt utveckla en första version av batteriövervakningsalgoritmen och den tillhörande mjukvaran.

Områden som behandlas i projektet är elektronikkonstruktion, analog design, mikrodator teknik, programmering, felsökning, test och verifiering.

1.3 Avgränsningar

Konstruktionen av modulen kommer att sträcka sig från att skapa kravspecifikation, designa hårdvara och utveckla mjukvara till att designa layout för kretskort. Tyngdpunkten ligger inte på layout av kretskort och batterikemi, även om en förståelse för dem är viktiga vid konstruktion av ett batteriövervakningssystem. Avslutningsvis är det tänkt att testa och verifiera systemet för att utvärdera i vilken grad prestanda och pålitlighet uppfyller projektmålen.

1.4 Mål

1.4.1 Grundläggande mål

- Spänningsmätning (Batterispänning inom $\pm 2.5 \%$)
- Strömmätning (Urladdningsström inom $\pm 5 \%$)
- Databehandling
- Seriell kommunikation
 - Status, Konfiguration (Nominell spänning & kapacitet, batterikemi), Sändning av data
- CAN kommunikation
 - Batteristatus, spänning och strömförbrukning
 - Exempelvis:
 - Batteri OK – Inga nödvändiga åtgärder
 - Låg batterinivå – Begränsa energiåtgången
 - Batteriet urladdat – Återladda batteriet, stäng av vissa förbrukare
 - Batteri slitet – Byt batteri
- Uppskattning av batterikapacitet inom $\pm 10 \%$
- Automatiska beslut från modulen om strömbegränsning och återuppladdning

1.4.2 Högre mål

- Spänningsmätning (Batterispänning inom $\pm 1 \%$)
- Strömmätning (Urladdningsström inom $\pm 2.5 \%$)
- Externt minne för nerladdning av flera körcykler, kompatibelt mot Excel/Access
- Övervakning och beräkning av batteriets åldrande
- Automatiska beslut från modulen om batteribyte
- Noggrannare uppskattning av batterikapaciteten, exempelvis $\pm 5 \%$
 - Hänsyn till icke-ideala faktorer (Beroende av batterikemi)
 - Efter förundersökning av de viktigaste icke-ideala faktorerna för batterikemin
 - Bland annat Peukert-effekten, åldrande eller körsträcka
- Uppskatta kvarvarande drifttid eller körsträcka (med hänsyn till Peukert-effekten)
- Uppskatta bland annat batteristatus, batterinivå, uppskattad drifttid, uppskattad tid till fulladdat batteri, aktuell batterispänning, genomflytande ström, cykelantal

2 TEKNISK BAKGRUND

2.1 Algoritmer och metoder

Urladdning och uppladdning av ett litiumjonbatteri utanför de av tillverkaren rekommenderade gränserna kan få förödande konsekvenser. I batterierna finns därför skyddskretsar som ser till att dessa gränser aldrig överskrids. Se avsnitt 2.3.6 : *Teknisk bakgrund: Litiumjonbatteriet: Krav på säkerhet*. Batteriövervakningssystemets huvudsyfte är därför att ge användaren indikation om batteriets laddningsnivå så att batteriets kapacitet utnyttjas maximalt. Batteriövervakningssystemet löser uppgiften genom att övervaka och kontrollera batteriets upp- och urladdningsprocess.^{1 (s. 2)} Detta kan göras genom att kontinuerligt mäta ström, spänning och temperatur. Dessa parametrar används sedan som indata till batteriövervakningsalgoritmen för att uppskatta de två parametrarna laddningsgrad (SoC), återsående körtid (t_r) och hälsostatus (SoH).^{1 (s. 3)}

State-of-Charge (SoC) anger hur stor del den aktuella laddningsnivån utgör av batteriets verkliga kapacitet (Q_{TOT}). State-of-Charge refereras här till som laddningsgrad. Förändringen av laddningsgrad är beroende av temperatur, urladdningshastighet och åldrande hos batteriet. Temperatur och urladdningshastighetens inverkan på kapaciteten går att kompensera för genom att implementera en uppslagstabell (eng. look-up table) som kompenserar för dessa effekter. Dess inverkan på batteriets åldrande är dock något mer komplicerat och går inte att kompensera för endast genom en uppslagstabell.^{1 (s. 3)}

För att uppskatta vilken effekt batteriets åldrande har på batteriövervakningssystemet används State-of-Health (SoH). State-of-Health refereras här till som hälsostatus. Batteriets hälsostatus är ett mått på hur stor del den verkliga kapaciteten (Q_{TOT}) utgör av den nominella kapaciteten (Q_{NOM}). Hälsostatusen hos ett batteri kommer att vara beroende av hur batteriet används. Batteriet slits till exempel olika beroende på hur stor ström som batteriet belastas med under ett bestämt tidsintervall och vid vilken laddningsgrad batteriet återuppladdas.^{1 (s. 4)}

Laddningsgrad och hälsostatus kan tillsammans med medelströmmen under körning användas till att beräkna kvarvarande körtid (t_r). Uppskattningen är beroende av om lasten är av ström- eller effekt-typ. Om lasten är av strömtyp uppskattas laddningsgraden lämpligen med beräkningar som involverar laddning (Coulomb eller Ah). Kvarvarande körtid erhålls då genom att dividera återstående laddning (Q_{AVA}) med den konsumerade strömmen. För laster av effekt-typ används lämpligen energi (J) istället vilket leder till att kvarvarande körtid erhålls genom att dividera tillgänglig energi med den konsumerade effekten.^{1 (s. 4)} Anledningen till detta är att en last av effekt-typ konsumerar en konstant effekt (endast beroende av användandet). Därmed är det den tillgängliga energin som är intressant.

Ett typiskt batteriövervakningssystem består av sensorer för spänning, ström och temperatur. Dessa storheter behöver sedan omvandlas till digitala värden för att mikroprocessorn ska kunna utföra de beräkningar som batteriövervakningsalgoritmen kräver. Spänningsmätning hör till den enklare typen av mätningar att utföra. Detta eftersom A/D-omvandlaren ofta arbetar med spänning som analog insignal.^{2 (s. 21)} Strömmätning är något mer avancerad, detta eftersom den kräver någon sorts

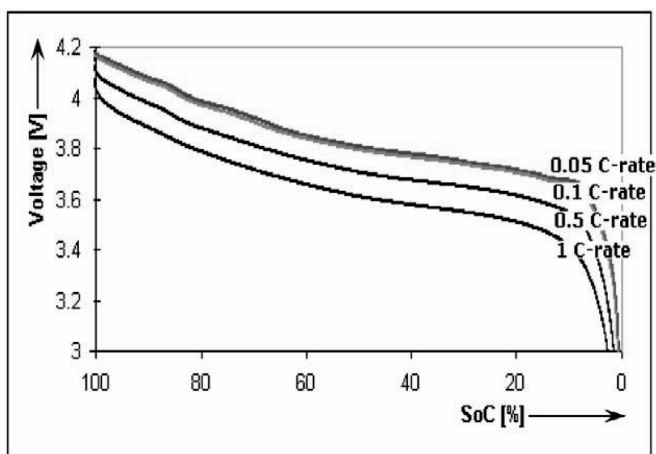
givare. Två vanliga strömgivare är strömshunt och halleffektsensorn. Gemensamt för dessa två är att utsignalen har storheten spänning. Se avsnitt 2.4 : *Teknisk bakgrund: Spännings- och strömmätningssmetoder*.

Det behövs algoritmer och metoder för att ett batteriövervakningssystem ska kunna göra en uppskattning av parametrar som laddningsgrad (SoC), återstående körtid (t_r) och hälsotillstånd (SoH) hos ett batteri. Det finns många olika metoder för att uppskatta dessa parametrar. Generellt kan metoderna delas upp i kategorierna *direkta mätningar*, *bokförande system* och *adaptiva system*.¹ (s. 25-26) Metoderna (och algoritmerna) är olika avancerade och har olika för- och nackdelar. Ett batteriövervakningssystem kan därför kombinera olika mätmetoder för att kompensera för enskilda metoders svagheter. En del av svårigheten i att utveckla batteriövervakningssystem ligger i oförutsägbarheten hos både batterier och användarmönster.

2.1.1 Direkta mätningar

Direkta mätningar (eng. Direct measurement) innefattar metoder som mäter någon storhet hos batteriet. Detta kan t.ex. vara spänning (V), impedans (Z) eller spänningens vilotid efter ett strömsteg (τ).¹ (s. 26) Mätningarna kan även kompenseras för temperatur (T). Genom att storheterna till viss del korrelerar mot batteriets laddningsgrad, kan denna på så sätt uppskattas. Den huvudsakliga fördelen med direkta mätningar är att mätningen kan göras direkt när ett batteri ansluts.¹ (s. 26) Detta betyder att urladdningshistoriken inte behöver vara känt.

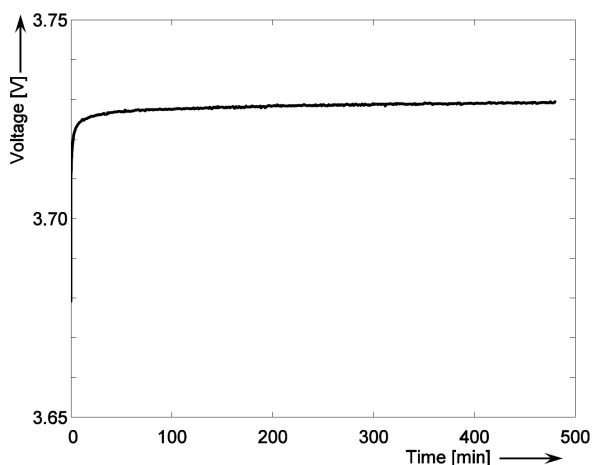
Det finns flera typer av mätningar som kan klassas som direkta mätningar. Mätning av ett batteris pol-spänning är en sådan. Mätningen är enkel att genomföra, men om urladdningsströmmen inte är känd och dessutom konstant, ger denna typ av mätning en dålig uppskattning av laddningsgraden.¹ (s. 26) Detta beror på att polspänningen inte bara varierar med laddningsgraden. I figuren nedan visas urladdningskaraktäristiken hos ett litiumjonbatteri vid olika normerade urladdningsströmmar. Figuren visar på att olika urladdningsströmmar starkt påverkar urladdningskaraktäristiken.



Figur 2.1: Batterispänningen varierar som en funktion av både laddningsgrad och urladdningsström. Källa: V. Pop, H.J. Bergveld, D. Danilov, P.P.L. Regtien and P.H.L. Notten.¹ (s. 27)

Om ett batteri tillåts vila, så att ingen ström flyter till eller från batteriet kommer polspänningen att närma sig EMF-spänningen.¹ (s. 27) Mätning av EMF-spänningen är ytterligare en typ av direkt mätning. EMF står för electromotive force, vilket på svenska kan översättas till elektromotorisk kraft. Denna är batteriets interna drivkraft för avgivande av energi till lasten. EMF-spänningen visar

sig korrelera väl mot ett litiumjonbatteris laddningsgrad.^{1 (s. 27)} Vidare ändras inte förhållandet nämnvärt av cykling och temperaturförändringar. Huvudproblemet med metoden är att det tar lång tid för pol-spänningen att närma sig EMF-spänningen. Speciellt när laddningsgraden är låg, vid låga temperaturer, eller efter en kraftig urladdningsström.^{1 (s. 65)} Figuren nedan visar hur pol-spänningen närmar sig EMF-spänningen.



Figur 2.2: Pol-spänningen närmar sig EMF-spänningen efter att en last om 0.1 C kopplats från batteriet. Källa: V. Pop, H.J. Bergveld, D. Danilov, P.P.L. Regtien and P.H.L. Notten.^{1 (s. 55)}

Det finns några tillvägagångssätt för att hantera fenomenet. Ett enkelt sätt är att helt enkelt vänta på att pol-spänningen ska närma sig EMF-spänningen. Detta löser emellertid problemet med att erhålla ett värde på EMF-spänningen, men erfordrar en väntetid om minst 30 minuter, gärna mer, för att erhålla ett bra värde.^{1 (s. 65)} I praktiken blir detta naturligtvis väldigt opraktiskt i många applikationer, eftersom batteriet måste vara i vila hela väntetiden, dvs. strömmen till och från batteriet måste vara noll under denna period.

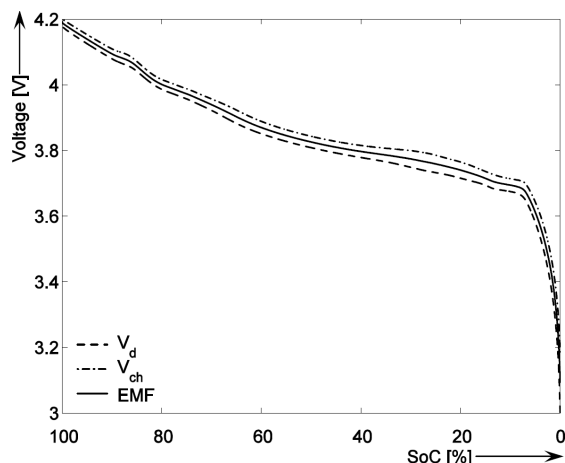
Ytterligare ett tillvägagångssätt för att uppskatta EMF-spänningen är spänningsuppskattning.^{1 (s. 70)} Metoden bygger på att ett värde på EMF-spänningen kan uppskattas innan pol-spänningen har närmat sig EMF-spänningen. Bland flera metoder för att uppskatta EMF-spänningen finns asymptot-metoden presenterad av Aylor et al. Denna bygger på att det går att uppskatta EMF-spänningen genom att placera en linjär asymptot mellan två samplingspunkter i ett logaritmerat tidsplan. Genom att läsa av den slutgiltiga polspänningen efter en förutbestämd tid erhålls sedan ett uppskattat värde av EMF-spänningen.^{1 (s. 71)}

EMF-metoden kräver givetvis att förhållandet mellan laddningsgrad (SoC) och EMF-spänning är väl känt. I praktiken används tre metoder för att korrelera SoC mot EMF. Dessa är *uppslagstabell*, *stegvis linjarisering* och *matematisk funktion*.^{1 (s. 29-30)}

Metoden *uppslagstabell* bygger på att en mängd av fasta värden för EMF är korrelerade mot värden för SoC. Storleken på uppslagstabellen begränsar den erhållna upplösningen. Metoden *linjarisering* miskar kravet på mängd sparad data genom att värden för SoC interpoleras mellan de sparade punkterna.

Fördelen är att noggrannare korrelation mellan EMF och SoC kan erhållas med mindre data. Metoden *matematisk funktion* bygger på att förhållandet mellan EMF och SoC är approximerat med en matematisk funktion. Genom att adaptivt anpassa funktionens parametrar kan metoden tillhandahålla det bästa resultatet för förhållandet mellan EMF och SoC.^{1 (s. 30)}

Figuren nedan visar den typiska EMF-SoC-karaktäristiken hos ett litiumjonbatteri.



Figur 2.3: EMF i förhållande till SoC hos ett litiumjonbatteri erhållet genom linjär interpolation av spänningen vid uppladdning och urladdning vid en ström om 0.05 C. Källa: V. Pop, H.J. Bergveld, D. Danilov, P.P.L. Regtion and P.H.L. Notten.^{1 (s. 64)}

Till direkta mätningar hör även metoden *impedansmätning*. Metoden bygger på att batteriets komplexa impedans mäts över ett frekvensområde. Detta kallas då elektrokemisk impedansspektroskopi (eng. Electrochemical Impedance Spectroscopy).^{1 (s. 30)} Impedansens beroende av frekvens kan då plottas i ett Bode- eller Nyqvist-diagram. Metoden lämpar sig dock inte så väl till att uppskatta laddningsgrad i portabla applikationer eftersom en signal med ett frekvenssvep måste appliceras till batteriet. Dessutom gäller att impedansens beroende av laddningsgrad är mindre än temperaturberoendet. Metoden lämpar sig bättre för att uppskatta batteriets hälsotillstånd.^{1 (s. 32)}

2.1.2 Bokförande system

Metoder som både mäter och integrerar strömmen för att uppskatta laddningsgraden tillhör kategorin *bokförande system* (eng. Book-keeping system). Dessa kan använda något som kallas Coulomb-räkning. Detta innebär att laddning till och från batteriet räknas.^{1 (s. 32)} Tillsammans med andra parametrar som självurladdning, temperatur, laddnings- och urladdningseffektivitet (även Peukert-effekten) och livslängd kan ett bokförande system uppskatta laddningsgraden.^{1 (s. 32)} Fördelen med Coulomb-räkning är att laddningsgrad kan uppskattas under upp- och urladdning av batteriet. Till nackdelarna hör att det ställs stora krav på strömmätningen i systemet. Eftersom metoden bygger på att ström integreras över tid gäller det att även små fel i mätningarna ackumuleras och orsakar stora fel i uppskattningen av batteriets laddningsgrad.^{1 (s. 33)}

2.1.3 Adaptiva system

Det som kännetecknar *adaptiva system* (eng. Adaptive system) är att de kan anpassa sina parametrar under cykling. Detta gör att de kan fortsätta att uppskatta laddningsgraden noggrant även när batteriet åldras, och dess karaktäristik ändras.^{1 (s. 7)} Adaptiva system använder sig av direkta mätningar, bokförande system eller en kombination av dessa. Implementationen kan bland annat göras med artificiella neurala nätverk (eng. Artificial Neural Network), suddig logik (eng. Fuzzy

Logic) och Kalman-filter.¹ Fördelen med adaptiva system är att de tar hänsyn till åldrande och generellt sett tillhör de noggrannaste systemen.^{1 (s. 41-42)} Till nackdelarna hör att de är komplexa att implementera. Artificiella neurala nätverk behöver mycket data för träning av systemet och är således dyrt att utveckla. Suddig logik kräver mycket arbetsminne för algoritmen. Kalman-filter är svåra att implementera om algoritmen ska ta hänsyn till icke-normala och icke-linjära faktorer.^{1 (s. 38)}

2.2 Fixpunktsaritmetik

I det här systemet används fixpunktsaritmetik. Detta innebär att decimaltal sparas med en heltalsdel (QI) och en decimaldel (QF) i ett heltal av en ordlängd $WL = QI + QF$. Fördelen med att spara decimaltal i heltalsvariabler är att beräkningar med heltal generellt kan göras mycket snabbare än med flyttal i majoriteten av de processorer som finns på marknaden.³ Orsaken till prestandavinsten är att många enklare processorer inte har någon inbyggd hårdvara för att göra beräkningar på flyttal. Prestandavinsten kommer dock med en kostnad. Eftersom decimalpunkten är fixt placerad blir det dynamiska området begränsat.

Ett tal sparad med fixpunktsaritmetik placerar en imaginär binal mellan heltalsdelen och decimaldelen. I detta system används uteslutande tal sparade med 8-bitars heltalsdel (QI) med tecken (s) och en 8-bitars decimaldel (QF) samt refereras till som ett fraktionellt tal. Heltalsdelen tolkas enligt tvåkomplementsaritmetik. Ordlengthen av detta fraktionella tal blir då 16 bitar och får följande representation vid användning av datatypen int. Notationen för det fraktionella talet kan illustreras med:

$$Q[QI].[QF] = Q[8].[8] = \\ = [s][b_6][b_5][b_4][b_3][b_2][b_1][b_0].[b_{-1}][b_{-2}][b_{-3}][b_{-4}][b_{-5}][b_{-6}][b_{-7}][b_{-8}] \quad \text{Formel 2.1.}$$

Talområdet av det fraktionella talet blir i huvudsak begränsat av antalet bitar i heltalsdelen (QI). För ett fraktionellt tal med tecken gäller det att talområdet blir begränsat till:

$$[-2^{QI-1}, 2^{QI-1}-2^{-QF}] \quad , \quad \text{Formel 2.2.}$$

där QI och QF är antalet bitar i heltals- respektive decimaldelen.^{3 (s. 11)}

Den undre gränsen kan härledas till att $QI - 1$ bitar finns tillgängliga för att representera negativa tal eftersom talet 0 lagras som ett positivt tal och en bit används som teckenbit. Den övre gränsen kan härledas av att $QI - 1$ bitar finns tillgängliga för att representera positiva tal samt att talet 0 behöver lagras. Detta skulle utan decimaldelen QF resultera i att $2^{QI-1} - 1$ är den övre gränsen dock ökar decimaldelen gränsen med det maximala värde decimaldelen kan anta vilket är $1 - 2^{QF-1}$. De fraktionella talen i detta system kan således anta värden i talområdet:

$$[-2^{8-1}, 2^{8-1}-2^{-8}] = [-128, 128 - 256^{-1}] \approx [-128, 127.996094] \quad \text{Formel 2.3.}$$

Även upplösningen hos det fraktionella talet blir begränsat. Upplösningen begränsas av antalet bitar i decimaldelen (QF). För ett fraktionellt tal gäller att upplösningen (ϵ) kan beräknas med sambandet:

$$\epsilon = \frac{1}{2^{QF}}, \text{ där } QF \text{ är antalet bitar i decimaldelen.}$$

Formel 2.4.

Upplösningen kan härledas till att decimaldelen helt enkelt delas upp i 2^{QF} steg. För de fraktionella talen i detta system blir upplösningen således:

$$\epsilon = \frac{1}{2^{QF}} = \frac{1}{2^8} = \frac{1}{256} \approx 0.003906$$

Formel 2.5.

En av fördelarna med fraktionella tal är att addition och subtraktion kan utföras precis som för heltal. Multiplikation och division är också enkla att utföra och kräver endast en skiftning av bitarna efter utförd operation. Likt andra operationer av binära tal gäller det att försäkra sig om att beräkningen av talen ryms inom talområdet. För mer information om fixpunktsaritmetik hänvisas till Fixed-Point Representation & Fractional Math, Erick L. Oberstar [Ref 3].

2.3 Litiumjonbatteriet

2.3.1 Bakgrund

Litiumjonbatterier används idag i majoriteten av alla portabla enheter. Exempel är bärbara datorer, mobiltelefoner, kameror och motordrivna verktyg. Batteritypen har även på senare tid börjat användas mer inom bilindustrin.⁷ Det första kommersiella litiumjonbatteriet såldes av Sony 1991 och den globala marknaden uppskattades 2010 att uppgå till över 10 miljarder dollar.⁷

Litiumjonbatteriet är ett underhållsfritt batteri, till skillnad från många andra batterikemier.⁹ Listan med fördelar kan göras lång. Några av dem är: mycket hög energidensitet (omkring 300 Wh/l eller 150 Wh/kg), den högsta elektrokemiska potentialen av de kända batterikemierna (genomsnitt 3.7 Volt), möjlighet till hög urladdningseffekt (upp till 2C), fritt från tungmetaller, hög cyklingstålighet, låg självurladdning, saknar minneseffekt, inget krav på vid vilken laddningsgrad batteriet bör laddas upp och att det kan återuppladdas till 80 % på under en timme.^{4,9}

Dessvärre har litiumjonbatteriet också några nackdelar. Till de mer omtalade ingår behovet av skyddskretsar för att hålla spänningsnivåer och strömmar inom säkra gränser, restriktioner vid transport samt att de är dyra att tillverka. De mindre omtalade nackdelarna är främst litiumjonbatteriets tendens att åldras, dvs. att dess kapacitet permanent sjunker, även utan att batteriet används. Ett fulladdat litiumjonbatteri tappar t.ex. 20 % av sin kapacitet efter ett år av lagring i rumstemperatur.⁵ Vid förvaring av batteriet i t.ex. en bärbar dator är förhållandena ännu mer ogynnsamma på grund av den högre temperaturen.

Litiumjon-kemin har en stor potential för framtida utveckling. Mycket pengar investeras världen över i forskning av nya anod- och katodmaterial hos litiumjonbatteriet. Till exempel forskare vid Stanford University har hittat ett sätt att markant öka kapaciteten i litiumjonbatterier. Genom att konstruera anoden, som vanligtvis består av grafit, med hjälp av nanorör i kisel har kapaciteten kunnat ökas 10 gånger.⁶

2.3.2 Uppbyggnad

Litiumjonbatteriet är ett sekundärbatteri. Det är uppbyggt av en positiv- och negativ elektrod samt en elektrolyt som finns i en separator mellan plattorna. Den positiva elektroden består av en kombination av metall tillsammans med oxid eller fosfat. Vanliga kombinationer är litium - kobolt - oxid (LiCoO_2), litium - mangan - oxid (LiMn_2O_4), litium - järn - fosfat (LiFePO_4) och litium - nickel - mangan - kobolt - oxid (LiNiMnCoO_2)^{7, 8}. Den negativa elektroden består oftast av någon form av kol, varav grafit (C_6) är vanligast.^{7, 8} Elektrolyten består av en blandning av flytande organiska lösningar och litiumsalter.⁷ Elektrolyten är fri från vatten eftersom litium reagerar kraftigt med vatten. Detta medger också att litiumjonbatterier har högre energidensitet än andra batterikemier.

Traditionella litiumjonbatterier är cylindriska celler med ett poröst membran, kallat separator, placerat i elektrolyten mellan den positiva och negativa elektroden. Separatorns uppgift är att förhindra kontakt mellan elektroderna och därmed att elektrisk ström leds mellan den positiva och negativa elektroden. För att batteriet ska fungera måste dessutom separatorn släppa igenom joner.

Skillnaden mellan litumpolymer- och litiumjonbatterier ligger enbart i elektrolyten.⁹ Litiumjonbatteriet använder sig av en flytande elektrolyt medan litumpolymerbatteriet använder sig av en torr fast polymerelektrolyt. Denna elektrolyt är en gel som bildas när separatorn kommer i kontakt med elektrolyten vid tillverkningen. På grund av detta kan litumpolymerbatterier tillverkas i många olika former litiumjonbatteriet som enbart tillverkas i solida cylindriska celle.¹⁰

2.3.3 Icke-ideala faktorer

Ofta används en modell med en vattenbehållare för att förklara hur ett batteri fungerar. Vattennivån representerar då hur mycket energi som finns lagrat kemiskt i batteriet. Flöden in och ut ur batteriet motsvarar ström in och ut ur batteriet. En sådan här modell fungerar som princip, men stämmer inte väl överens med batterier i verkligheten, speciellt vid stora strömmar i förhållande i batteriets kapacitet.

Det finns ett antal olika faktorer som gör att tillgänglig energi i ett batteri inte är lika mycket som den energi som tillförs vid uppladdningen.¹¹ Några av dessa är laddningsverkningsgrad, urladdningshastighet och -pulsqvot, spänningskaraktäristik, självurladdning och inre resistans. Dessa faktorer påverkas i sin tur av bland annat ålder, slitage och temperatur.

För att bättre uppskatta kapaciteten hos ett batteri måste hänsyn tas till ett antal av dessa faktorer. Ett sätt att göra det är att skapa modeller av ett icke-idealt batteri. Detta är mycket svårt eftersom det inte är helt känt hur dessa faktorer påverkar kapaciteten.¹¹

W. Peukert var en tysk vetenskapsman som 1897 presenterade en formel som visar hur kapaciteten (C_{Peukert}), vid en urladdningstid (t) om en timme, i ett bly-syrabatteri beror av urladdningsströmmen och en konstant (k).¹² Formeln kom att kallas Peukerts lag och visar att tillgänglig kapacitet sjunker exponentiellt med urladdningsströmmens (I) storlek. Konstanten i formeln varierar beroende på batterityp och ålder. Ett värde nära 1 motsvarar ett nästan idealt batteri medan högre värden motsvarar mindre effektiva batterier. Nedan presenteras Peukerts formel.

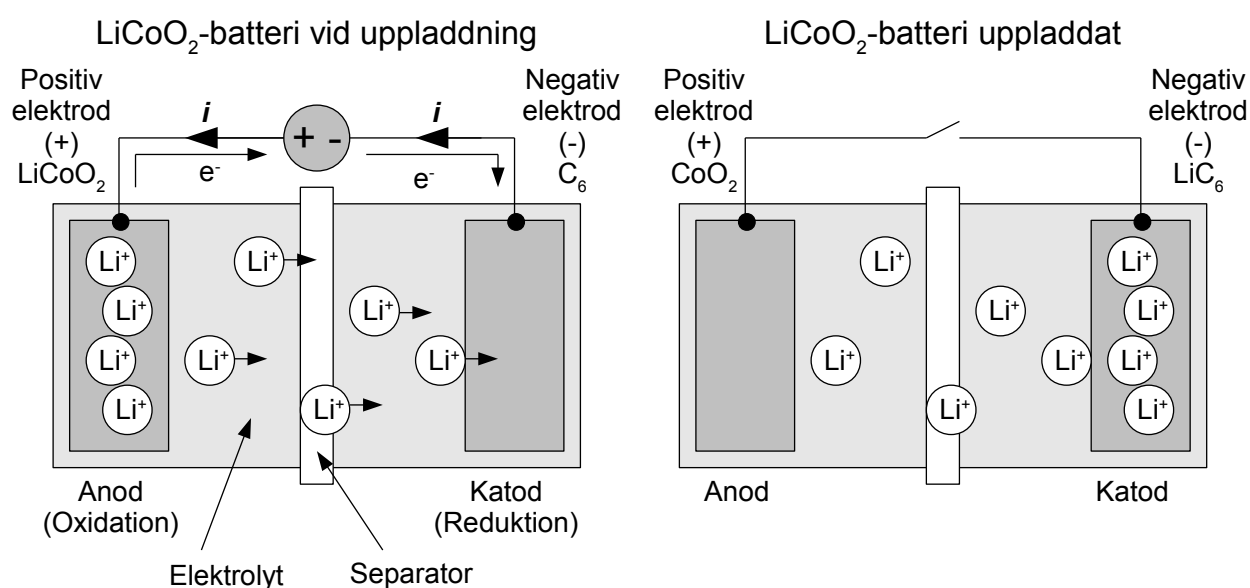
$$C_{\text{Peukert}} = I^k t$$

Formel 2.6.

2.3.4 Funktion

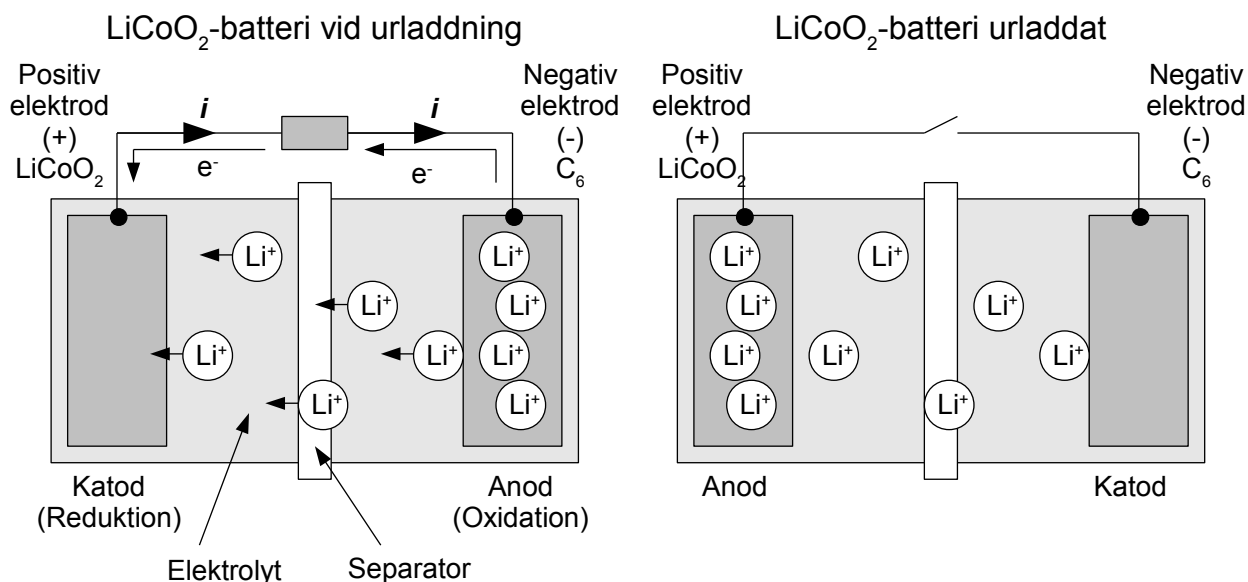
Det första kommersiella litiumjonbatterierna var av typen litium-kobolt-oxid (LiCoO_2).⁷ Namnet anger vilka ämnen som finns i den positiva elektroden. Den negativa elektroden består vanligtvis av grafit (C_6). En elektrolyt finns placerad mellan elektroderna. Dess uppgift är att möjliggöra transport av laddningsbärare, joner.

När ett nyttillverkat batteri för första gången laddas upp tillförs en ström till den positiva elektroden. Elektroner drivs således från den positiva till den negativa elektroden. För att jämvikt skall uppnås leds positivt laddade litiumjoner (Li^+), katjoner, över till den negativa elektroden av grafit, via elektrolyten. Processen kallas interkalering (eng. intercalation) och innebär att litiumjoner skjuts in i grafitens lager.⁷ Vid laddning undergår den positiva elektroden oxidation (avgivande av elektroner) och är därmed anod. Den negativa elektroden undergår samtidigt reduktion (upptagande av elektroner) och är därmed katod. I ett fulladdat batteri finns alltså en mängd litiumjoner interkalerade i grafiten.



Figur 2.4: Den grundläggande funktionen för ett LiCoO_2 -batteri vid uppladdning.

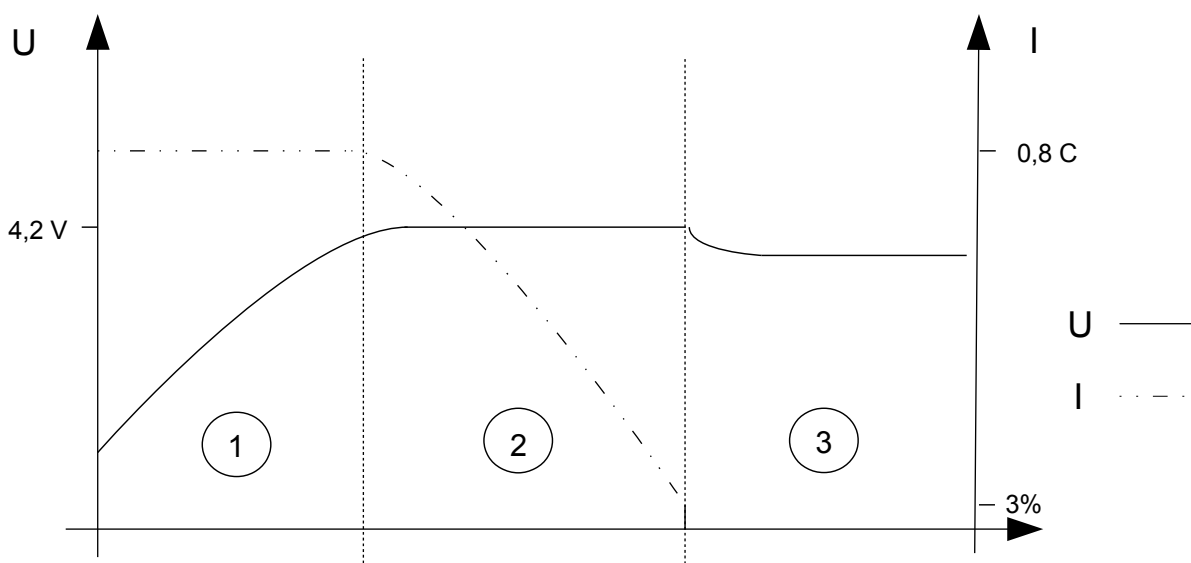
Vid urladdning av ett batteri sker det omvända. En ström drivs från den positiva elektroden till den negativa. Elektroner drivs alltså från den negativa elektroden till den positiva. För att jämvikt skall uppnås leds positivt laddade litiumjoner över via elektrolyten till den positiva elektroden. Processen kallas deinterkalering (eng. deintercalation) och innebär att litiumjoner lämnar grafiten.⁷ Vid urladdning undergår den positiva elektroden reduktion (upptagande av elektroner) och är därmed katod. Den negativa elektroden undergår samtidigt oxidation (avgivande av elektroner) och är därmed anod.



Figur 2.5: Den grundläggande funktionen för ett LiCoO₂-batteri vid urladdning.

2.3.5 Laddning av litiumjonbatterier

Uppladdning av ett litiumjonbatteri enligt CCCV-metoden (efter eng. Constant-Current-Constant-Voltage) kan delas in i tre faser.^{1 (s. 85)} Under den första fasen sker uppladdningen under strömbegränsning. När spänningen nått den maximala spänningen om 4,2 Volt påbörjas fas 2, under vilken spänningen på 4,2 Volt bibehålls medan strömmen sjunker. När strömmen underskrider 3% av sitt ursprungsvärde inleds fas 3 under vilken laddning av batteriet avslutas, batteriet är nu fulladdat. Ibland har laddaren även en fjärde fas då 4,2 Volt appliceras till batteriet varpå man återigen inväntar att strömmen når 3% av sitt ursprungsvärde.¹³ En typisk laddningscykel som innefattar alla fyra faserna tar ca 3 timmar med en strömnivå om 0,8 C vilket är den strömnivå som rekommenderas bland tillverkare av litiumjonbatterier.¹³ Ibland används en snabbare metod att ladda batteriet där endast fas 1 används. När spänningen över polerna nått 4,2 Volt anses batteriet vara fulladdat. Denna metod ger ett till 85% uppladdat batteri.¹³



Figur 2.6: Illustration över de tillstånd som laddning av ett litiumjonbatteri innefattar.

Laddning av ett litiumjonbatteri över spänningen 4,2 Volt leder till att litium börjar att bildas vid den negativa elektroden (katoden) samtidigt som det bildas koldioxid vid den positiva elektroden (anoden). Tillväxten av litium bildar dendriter vid den negativa elektroden. Om överladdningen tillåts fortskrida kan dessa penetrera separatorn och kortsluta batteriet internt.^{7 (s. 181)} En ökning av mängden koldioxid vid den positiva elektroden leder till att trycket och temperaturen ökar i batteriet vilket kan få förödande konsekvenser.¹³

Även överurladdning av batteriet är skadligt. Om cellspänningen i batteriet underskrider 1,5 Volt under en längre tid än en vecka kan detta resultera i att de litium dendriter som bildats vid anoden bryts loss.^{7 (s. 181)} Förutom att batteriet kapacitet minskar kan detta leda till att batteriet kortsluts internt.¹³

2.3.6 Krav på säkerhet

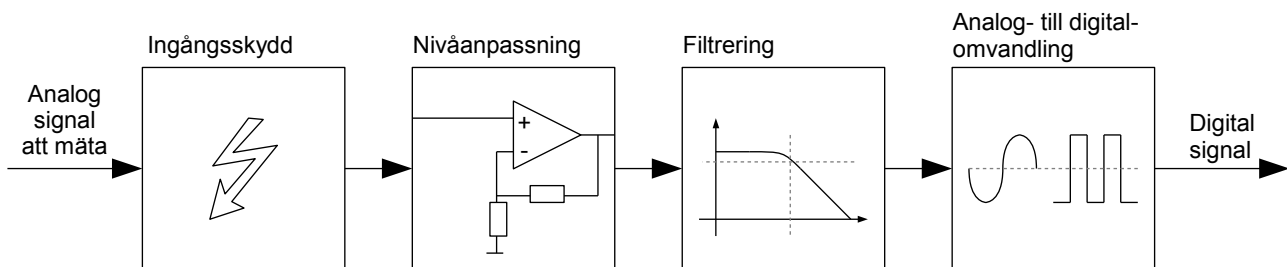
Litiumjonbatterier är känsliga för både över- och överurladdning. Skulle ett litiumjonbatteri ändå utsättas för detta kan följden bli en förhöjd temperatur i batteriet, vilket kan leda till att batteriet fallerar under kraftig energiutveckling. För att undvika detta finns det i varje litiumjoncell en inbyggd PTC-resistor som kraftigt begränsar strömmen vid hög temperaturutveckling. I cellen finns dessutom en trycksensor som bryter strömmen då trycket i cellen överskrider 1000 kPa. Skulle trycket stiga ytterligare finns det även en säkerhetsventil som släpper ut koldioxid ur cellen för att sänka trycket i cellen och lugna förbränningsförloppet.¹⁴ Utan dessa säkerhetsåtgärder skulle litiumjonbatterier falla mer frekvent än vad de gör idag. Den explosion som ett fallerande batteri utan skyddskrets skulle kunna utlösa hade gjort batteriet farligt att använda. En stor del i att kommersialisera litiumjonbatteriet har därför bestått i att förse det med bra säkerhetskretsar.¹⁵

Förutom dessa säkerhetsanordningar som litiumjonbatterierna är försedda med på cell-nivå är de flesta kommersiella batterier även försedda med en extern skyddskrets. Denna ser till att batteriets spänning alltid befinner sig i intervallet 3,0 – 4,2 Volt.¹³ Detta för att undvika över- och överurladdning. Under långvarig lagring av batteriet kan självurladdning medföra att cellspänningen i batteriet understiger 2,7 Volt. Skulle detta ske kan batteriet inte längre laddas upp i de flesta batteriladdare och batteriet sägs vara i vila.¹⁴

Trots en fungerande skyddskrets kan det vid sällsynta tillfällen hända att det uppstår en värmereaktion i litiumjonbatteriet. Det beror på att det kan finnas små metallpartiklar i battericellen. Om många små metallpartiklar samlas på ett ställe går det en liten ström mellan elektroderna vilket leder till att batteriet hettas upp. Eftersom att skyddskretsen är oförmögen att förhindra en eskalerande värmereaktion när den väl har startat leder detta till att batteriet fallerar. För att förhindra att värmereaktionen sprider sig till närliggande celler bör battericeller vara väl avdelade ifrån varandra.¹⁵

2.4 Spännings- och strömmätningssystem

Ett digitalt elektriskt mätsystem kan beskrivas som ett antal block i en kedja. Varje block har en specifik uppgift som berör anpassning och förändring av signalen. Olika mätsystem kräver olika uppsättningar av dessa. Om systemet t.ex. behöver mäta en fysikalisk storhet behövs en *givare* först i blockschemat. Ofta är det möjligt att signalen in till ett mätsystem överlagras med skadliga spänningsimpulser, som t.ex. från en elektrostatisk urladdning. För att inte mätsystemets komponenter ska ta skada av dessa behövs ett *ingångsskydd*. Det är viktigt att detta klarar att dämpa den skadliga signalen samtidigt som normala (förväntade) signaler inte distorderas. Sällan kan rätt signalnivå erhållas direkt från mätobjektet. Därför behövs ofta någon form av *nivåanpassning* som t.ex. förstärkning av signalen från en givare, eller dämpning av signalen från en högspänningskälla. Vidare måste signalens bandbredd begränsas så att inte vikning uppstår vid sampling. Detta görs genom *filtrering* av signalen genom ett lågpasfilter. För att signalen sedan ska kunna tolkas av en dator krävs att den omvandlas i en *analog till digitalomvandlare*. Fler steg som kan innefattas i ett digitalt elektriskt mätsystem är bland annat signal- och databehandling.¹⁶



Figur 2.7: Förenklat blockschema över ett digitalt elektriskt mätsystem.

Spänningsmätning hör till de enklare signaler att mäta med ett digitalt elektriskt mätsystem. Detta grundar sig i att systemet redan är elektriskt. Enkla system av denna typen behöver *ingångsskydd*, *nivåanpassning*, *filtrering* och *analog till digitalomvandling*. *Nivåanpassningen* i mätsystemet behöver anpassa insignalens förväntade område så att analog till digital-omvandlarens område används så väl som möjligt. Generellt kan sägas att högre spänningar än analog till digitalomvandlarens referensspänning, behöver spänningsdelas. Detta kan göras på olika sätt, t.ex. med resistorer, operationsförstärkare eller transformator (vid växelspänning). Omvänt gäller vid lägre spänningar än analog till digitalomvandlarens referensspänning där spänningen behöver förstärkas. Detta går t.ex. att göra med hjälp av operationsförstärkare eller transformatorer (vid växelspänning). Spänningsmätning blir svårare vid väldigt höga eller låga spänningar relativt referensspänningen.

Strömmätning är mättekniskt mer avancerat än spänningsmätning. Orsaken är att någon typ av *givare* är nödvändig för att få en spänning som är proportionell mot en ström. Det finns tre huvudtyper av strömgivare, shunt, Halleffekt-sensorer med återkoppling samt Halleffekt-sensor utan återkoppling.¹⁷

En shunt är ett resistivt element som utnyttjar Ohms lag för att ge en spänning proportionell mot strömmen. Shunten är den noggrannaste och mest linjära typen av strömsensor.¹⁸ Dock har shunten ett antal nackdelar. Den kanske största nackdelen är att det blir ett spänningsfall över shunten vid höga strömmar. Detta kan skapa problem med förlusteffekter, ty effekt är produkten av spänning och ström. Vidare ökas källresistansen i kretsen. Shuntar kan även ha problem med temperaturgradienter där termoelektriska spänningar kan störa mätningar av låg ström.¹⁷ Det

dynamiska området för många shuntar är omkring 1000:1.



Figur 2 .8: En typisk shunt godkänd för 50 Ampere.¹⁹

Halleffekt-sensorer utnyttjar som namnet antyder Hall-effekten för att mäta ström. Sensorerna finns både med och utan återkoppling. Principen för en Halleffekt-sensor är att den ledare, genom vilken ström önskas mätas, placeras i mitten av en järnkärna. I järnkärnan finns ett gap där magnetfältet kan mätas med hjälp av en Halleffekt-sensor. De återkopplade sensorerna använder en lindning runt järnkärnan för att motkompensera magnetfältet och på detta sätt uppnå högre noggrannhet. Förstärkningen hos en Halleffekt-sensor utan återkoppling påverkas mer av temperaturen. Fördelarna med Halleffekt-sensorer är bland annat galvanisk isolation, låg förlusteffekt samt att ledaren genom vilken strömmen ska mätas, inte behöver delas. Dock har Halleffekt-sensorn också några nackdelar. Bland annat har de problem med offset-strömmar.¹⁷ Detta beror på att järnkärnan behåller en liten del av magnetfältet efter att ha blivit kraftigt magnetiserad, en så kallat hysteres. Vidare är det dynamiska området ofta begränsat till 200:1.



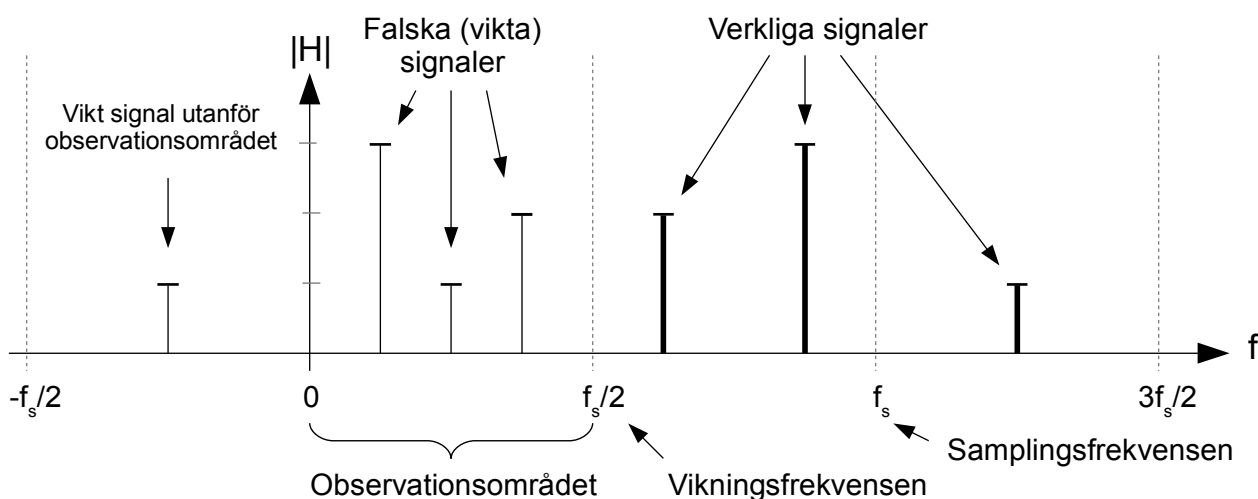
Figur 2 .9: En typisk Halleffekt-sensor godkänd för 60 Ampere.²⁰

2.5 Sampling, samplingsteoremet och antivikningsfilter

De storheter som ett digitalt mätsystem måste kunna mäta är väldigt ofta analoga och varierar med tiden. För elektriska givare gäller att en elektrisk signal beror av en fysikalisk storhet.²¹ För att en dator ska kunna tolka den elektriska signalen måste denna mätas. Detta kan göras på många olika sätt, gemensamt för dem är att de innefattar sampling av den analoga signalen och därefter en konvertering av denna till ett digitalt värde.

Sampling innebär att ett ögonblicksvärde av en signal tas. Flera samplingspunkter måste göras för att kunna följa variationer av signalen i tiden. Frekvensen av dessa samplingspunkter kallas samplingsfrekvensen, och är ofta konstant i ett mätsystem. Eftersom tidsintervallet för dessa samplingspunkter i praktiken alltid är ändligt går det att anta att signalen utanför detta intervall är periodisk. Därmed går det att se samlingsmängden som en periodisk funktion. Enligt Joseph Fourier gäller då att varje kontinuerlig funktion kan skrivas som summan av sinusfunktioner med varierande frekvens och amplitud. För att kunna återskapa signalen korrekt måste Nyquist-Shannons samplingsteorem vara uppfyllt.²² Samplingsteoremet säger att samplingsfrekvensen måste vara mer än dubbelt så hög som den högsta frekvenskomponent i signalen som samplas, för att denna korrekt ska kunna återskapas.

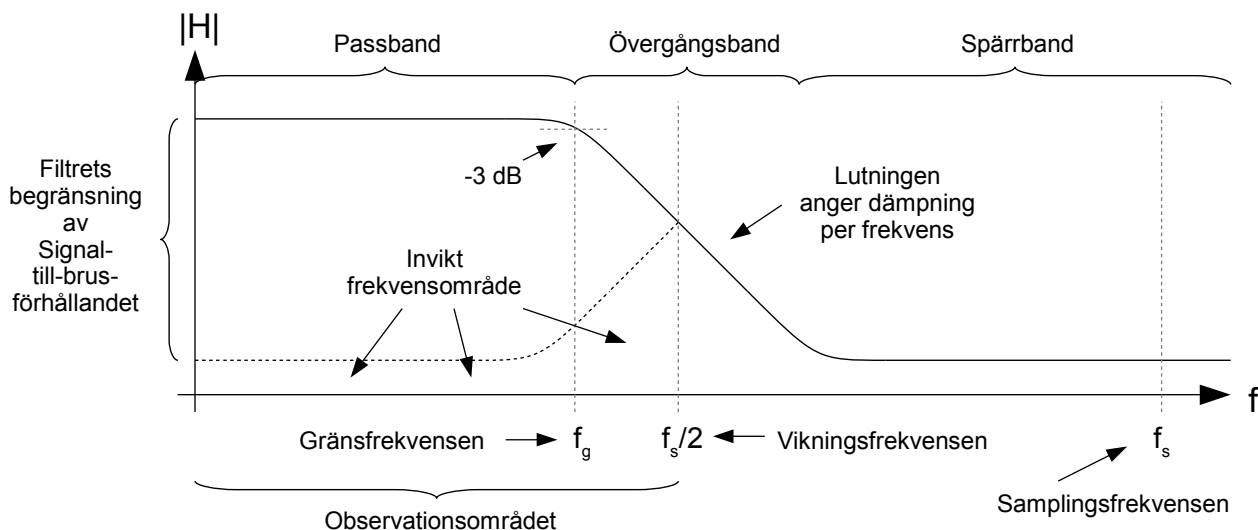
Om samplingsteoremet inte är uppfyllt sker något som kallas vikning (även kallat aliasing).¹⁶ Detta kan till exempel vara fallet med en signal som innehåller överlagrade störningar och brus, vilka oavsiktligt består av frekvenskomponenter över halva samplingsfrekvensen. I praktiken är detta mer eller mindre fallet för alla signaler. Vikning innebär att frekvenskomponenter över halva samplingsfrekvensen "viks" runt denna, tills att de framträder i intervallet noll till halva samplingsfrekvensen, det vill säga observationsområdet. Fenomenet gör att det blir omöjligt att skilja på frekvenskomponenter från observationsområdet och de över vikningsfrekvensen.



Figur 2.10: Falska signaler uppträder i observationsområdet på grund av vikning.

Problemet med vikning går till viss del att åtgärda genom att låta signalen som ska samplas passera genom ett filter, så kallat antivikningsfilter.²³ Detta filter är ett lågpasfilter med uppgift att släppa igenom signaler (läs frekvenskomponenter) i observationsområdet, samtidigt som frekvenskomponenter över vikningsfrekvensen spärras.

Flera parametrar ligger till grund för dimensioneringen av lågpåssfiltret. Det handlar ofta om en kompromiss mellan krav på signal-till-brusförhållande (eng. signal-to-noise ratio), önskad högsta frekvens av intresse (bandbredd), tillgänglig högsta samplingsfrekvens och kostnad för filtret.²⁴ Den grundläggande parametern som måste väljas är den gränshfrekvens filtret ska ha, det vill säga vilken frekvens som är den högsta av intresse. Genom val av önskad samplingsfrekvens och signal-till-brusförhållande (det vill säga filtrets dämpning) hos den resulterande signalen kan krav på filtrets prestanda beräknas. Värt att notera är att det slutgiltiga signal-till-brusförhållandet hos systemet bestäms av många faktorer och att antivikningsfiltret endast sätter en övre gräns på detta. Detta beror på att problemet med viking inte går att åtgärda fullt ut. Frekvenskomponenter över vikningsfrekvensen dämpas endast, av antivikningsfiltret. På så sätt kommer dessa till viss del alltid med, eftersom dessa viks in till observationsområdet, som illustrerat av figuren nedan.



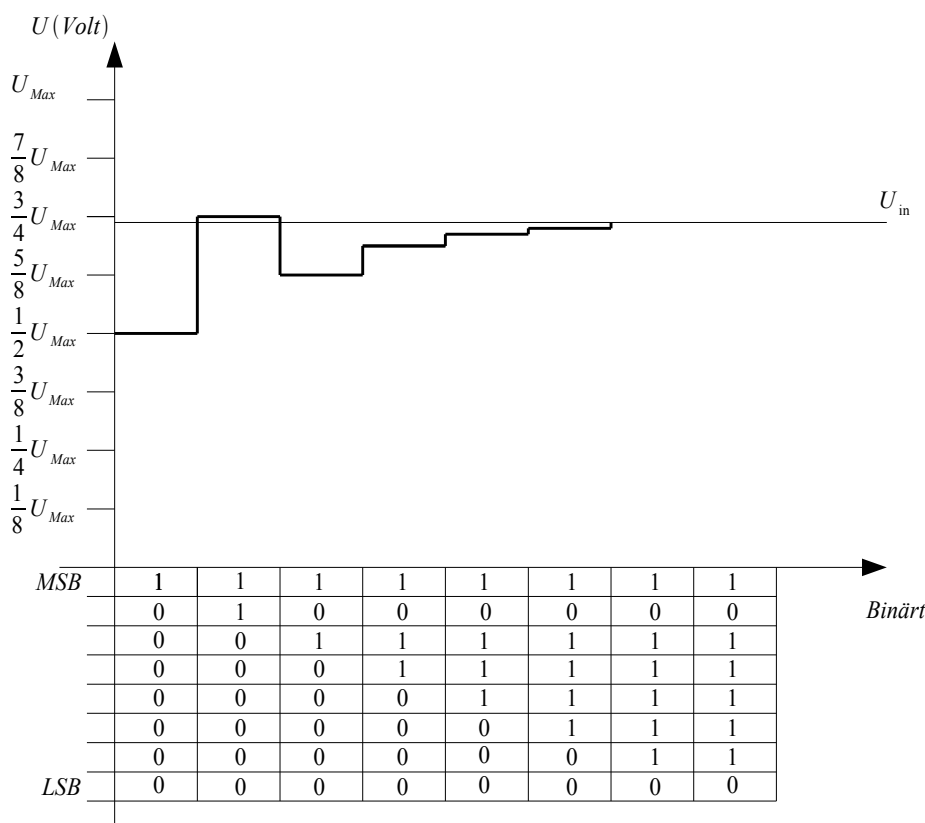
Figur 2.11: Förenklad bild över frekvenskaraktärstiken hos en antivikningsfilter.²⁴

Ett digitalt mätsystem kräver noggrann utvärdering och dimensionering av antivikningsfilter för att undvika problem med viking.

2.6 A/D-omvandling

I mikrodatorer sker oftast A/D-omvandling med en metod som kallas successiva approximationer. Metoden är uppbyggd kring en D/A-omvandlare och en komparator. En version av successiva approximationer är den så kallade räknande A/D-omvandlaren. Vid A/D-omvandling jämförs det analoga värdet med ett D/A-omvandlat värde som räknas upp via ett register. När komparatorn slår om från hög till låg är A/D-omvandlingen klar och det digitala värdet som motsvarar den analoga storheten kan avläsas i registret.²⁵ Nackdelen med metoden är att omvandlingen tar lång tid. Att A/D-omvandla ett 8-bitars värde kan ta ända upp till 256 klockcykler. I och med att det inte går att veta hur stort det analoga värdet är innan A/D-omvandlingen har ägt rum uppskattas alla A/D-omvandlingar att ta 256 klockcykler.²⁵ För att lösa problemet har ett smartare sätt att bestämma bitvärdena vid A/D-omvandling utformats.

Den smartare versionen bygger på att vid A/D-omvandling sätts först den mest signifikanta biten i registret till 1 medan de övriga är 0. Om den resulterande spänningen är lägre än det analoga värdet som ska bestämmas får 1:an vara kvar, i annat fall sätts biten till 0. A/D-omvandlingen sätter sedan nästa bit till 1, låter den vara kvar om värdet ut från D/A-omvandlaren ger ett värde lägre än det analoga värdet, annars 0.^{26(s.123-124)} På detta sätt bestäms det digitala värdet på ett 8 bitars register på endast 8 klockcykler att jämföra med de 256 som krävs med en räknande A/D-omvandlare.²⁵



Figur 2.12: Diagrammet visar funktionen hos en A/D-omvandling med successiva approximationer.

Ett problem med successiva approximationer är att det digitala värdet alltid kommer att representera ett värde lika med eller mindre än det analoga värdet. Detta kan ibland resultera i att det digitala värdets fel blir upp till värdet av den minst signifikanta biten. Genom att lägga till värdet av den

minst signifikanta biten då det digitala värdet skiljer mer än halva värdet av den minst signifikanta biten kan det maximala felet begränsas till hälften av den minst signifikanta biten.^{26 (s.126)}

2.7 Icke ideala faktorer hos operationsförstärkare

Vid beräkningar på operationsförstärkare är det vanligt att betrakta förstärkaren som ideal. Detta innebär att den antas ha oändlig inresistans, ingen utresistans och en oändlig förstärkning. Vid beräkning stämmer ideala förstärkarmodellen oftast väl överens med verkligheten. De icke ideala faktorerna har i de flesta fall mycket liten påverkan.^{2 (s.46)} Men i applikationer där differential mode-spänningen är låg kommer offset-spänningen att spela in, i applikationer där common mode-spänningen är hög kommer förstärkarens CMRR att spela in och i applikationer med stort utspänningssving och/eller hög frekvens kommer slewrate eller förstärkningsbandbreddsprodukten att spela in. Operationsförstärkaren kan dessutom inte ge en utspänning över matningsspänningen. I de flesta förstärkare ligger den maximala utspänningen något under matningsspänningen. Om det ändå skulle behövas en utspänning lika stor som matningsspänningen finns det operationsförstärkare som är speciellt framtagna för detta ändamål, så kallade rail to rail-förstärkare.^{2 (s.64)}

Offsetspänningar uppstår då ingångsteget till operationsförstärkaren inte är perfekt balanserat. Detta leder till att förstärkaren adderar eller drar ifrån en liten spänning på det önskade värdet på ingången.^{2 (s.64-65)} På en del förstärkare finns möjlighet att koppla in en potentiometer för att justera offsetspänningen, i annat fall finns det möjligheter att sköta denna kalibrering via en extern krets.^{2 (s.66)}

I applikationer med höga krav på noggrannhet kan en så kallad Chopper-förstärkare användas.²⁷ I Chopper-förstärkaren finns det fyra switchar som alternerar förstärkaren mellan två lägen. I det ena läget lagras offset-spänningen på en kondensator och i det andra läget adderas offset-spänningen till den negativa matningsspänningen på förstärkaren. Resultatet blir att spänningsintervallet på utgången förskjuts med ett värde motsvarande offset-spänningen.²⁸ Eftersom offset-spänningen mäts kontinuerligt blir förstärkaren inte lika känslig för temperaturändringar som en vanlig operationsförstärkare.²⁸

En annan sak som påverkar operationsförstärkarens noggrannhet är förstärkarens förmåga att undertrycka common mode-spänning. Detta är speciellt viktigt då två signaler med hög spänning ska jämföras. Ett mått på hur bra en förstärkare är på att undertrycka common mode-spänningar ges av common mode-rejection-ratio (CMRR).^{2 (s.70)}

Utresistansen hos en operationsförstärkare brukar antas vara noll vid typiska beräkningar. Ett antagande som fungerar i de flesta applikationer. Om lasten till förstärkaren är för stor kommer förstärkaren att få svårt att driva en tillräckligt stor ström för att bibehålla spänningen på utgången. Vid sådana tillfällen är det nödvändigt att höja resistansen hos lasten. Många operationsförstärkare har idag inbyggd strömbegränsning för att skydda mot att komponenter brinner upp vid höga strömmar.^{2 (s.73)}

När operationsförstärkaren används vid högre frekvenser kommer även dess förmåga att snabbt ändra utspänningen att spela roll. Denna egenskap anges i datablad som förstärkarens slew rate. Att tänka på vid sådana applikationer är att utspänningssvinget kommer att ha inverkan på förstärkarens övre gränsfrekvens.^{2 (s.74-75)}

2.8 Kommunikation

2.8.1 CAN

CAN står för Controller Area Network och är ett protokoll utvecklat för användning inom fordonsindustrin. Protokollet skapades i mitten av 80-talet av det tyska företaget Robert Bosch GmbH, mer känt som Bosch. Syftet med CAN är att möjliggöra robust seriell kommunikation mellan mikrodatorer (noder) via en gemensam buss. CAN är också tänkt att göra bilar mer tillförlitliga, säkra och bränslesnåla samtidigt som mängden kablage och dess komplexitet kan minska.²⁹ CAN används också inom industriell automation, medicinsk utrustning och testutrustning.

CAN är standardiserat av International Organization for Standardization (ISO) och finns i två versioner. Standarden för långsammare kommunikation heter ISO11519, samt standarden för snabbare kommunikation heter ISO11898.

Protokollet använder sig av åtkomstmetoden CSMA/CD vilket står för *Carrier Sense Multiple Access with Collision Detection*. *Carrier Sense* innebär att alla noder måste vänta på att den gemensamma bussen har varit ledig under en viss tid innan sändning får påbörjas. *Multiple Access* innebär att när en nod har detekterat att bussen är ledig, har den lika stor möjlighet att börja sända som alla andra noder. Detta eftersom CAN bygger på att alla noder på bussen måste tillåtas ta del av samma information vid varje tidpunkt (vilket också begränsar den fysiska utbredningen hos nätet). Den sista delen *Carrier Sense* i åtkomstmetoden står för att en nod i ett CAN-system använder något som kallas arbitrering. Detta möjliggör att en nod kan detektera om någon annan nod sänder på bussen samtidigt, samt beroende på dess prioritet eventuellt avbryta sin sändning.

Arbitreringsmekanismen i CAN-protokollet är icke-destruktiv och bygger på att varje CAN-nod har en unikt arbitreringsidentitet. Denna är ett unikt värde som explicit bestämmer CAN-nodens prioritet. När en nod vill sända ett meddelande lyssnar denna först på bussen och väntar tills den har varit ledig en viss tid. Efter att den detekterat en ledig buss börjar den sända sin arbitreringsidentitet samtidigt som den lyssnar på bussen. Genom att protokollet definierar ettor som recessiva och nollor som dominanta, kommer den nod med lägst arbitreringsidentitet (och därmed högst prioritet) att vinna arbitreringen. Den eller de noder med lägre prioritet (högre arbitreringsidentitet) slutar direkt att sända när de inte läser samma arbitreringsidentitet på bussen som de sänder ut.

CAN-protokollet är ett meddelandebaserat protokoll som bygger på att alla noder på bussen tar del av samma information. Varje nod kan sedan med hjälp av ett filter välja att ta eller inte ta emot ett meddelande. Protokollet definierar fyra olika typer av meddelanden, kallade *frames*. Normalt sett skickas meddelanden med något som kallas *Data Frame*. Dessa finns i två varianter beroende på hur många bitar arbitreringsidentiteten består av. *Standard Frame* använder 11 bitar, medan *Extended Frame* använder 29 bitar. Båda består av arbitreringsidentitet, kontrollfält, datafält, checksummafält, bekräftelsefält och meddelandeslut. Det finns även möjlighet för en nod att efterfråga ett meddelande från en annan nod. Denna typ av meddelande kallas *Remote Transmit Request Frame* (RTR). Vidare finns två till typer av meddelanden för hantering av fel. Dessa är *Error Frame* samt *Overload Frame*.²⁹

Robustheten i CAN-kommunikation bygger på ett antal olika funktioner. Dels kan en nod detektera

olika fel. Dessa är checksummafel (*CRC Error*), bekräftelsefel (*Acknowledge Error*), formatfel (*Form Error*), bitfel (*Bit Error*) samt bitstuffel (*Stuff Error*). Dels kan en nod, vid detektering av något av dessa fel, också ta beslut om att byta operationsmod. Normalt arbetar en CAN-nod med moden fel-aktiv (*Error-Active*), vilket innebär att en nod kan skicka och ta emot meddelanden. Vid upprepade fel kan noden gå över till att arbeta i fel-passivt läge (*Error-Passive*) eller nedkopplat läge (*Bus-Off*). Detta garanterar att en fallerande nod inte kan ockupera all bandbredd på bussen och kallas felinneslutning (*Fault Confinement*).

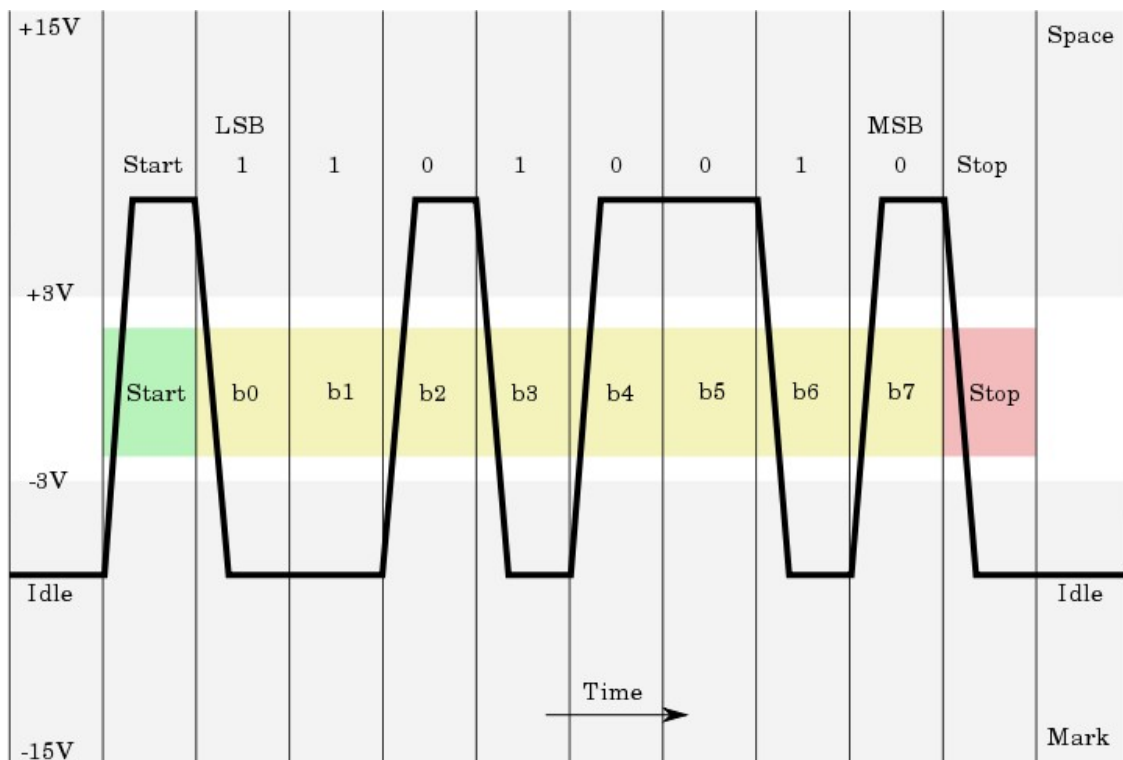
2.8.2 RS-232

Ett av de vanligaste protokollen för dataöverföring är det så kallade RS-232-protokollet.^{30 (s.10)} Enligt protokollet representeras en 1:a av spänningsnivåer mellan -3 och -15 volt och en 0:a av spänningsnivåer mellan +3 och +15 Volt. Övriga spänningar ignoreras av protokollet.^{30 (s. 11)}

Protokollet finns i två utförande, dels i fullt utförande med 25-polers DSUB-kontakt och dels i en nerbantad version med 9 polers DSUB kontakt. Skillnaden mellan de båda är framförallt att 25-polers varianten har pinnar för att sända och ta emot klockan vilket ger möjlighet att använda protokollet vid synkron överföring. Annars används protokollet främst vid asynkron dataöverföring via en 9-polig DSUB-kontakt av typ DE-9.^{30 (s. 12)}

I och med att protokollet används vid asynkron överföring finns behov av synkronisering mellan sändare och mottagare. Detta har lösts genom att varje meddelande föregås av en startbit som mottagarens klocka kan trigga på, därefter sker överföringen med en på förhand bestämd överföringshastighet, så kallad baud rate. Ett dataord kan bestå av mellan fem och åtta bitar med en eller flera påföljande stoppbitar. Eventuellt kan en paritetsbit läggas till för att detektera eventuella felaktigheter i överföringen. Det går bland annat att välja mellan udda, jämn eller ingen paritetsbit. Vid udda paritet sätts paritetsbiten till 0 om antalet ettor i dataordet är udda, vid jämn paritet sätts paritetsbiten till 0 om antal ettor i dataordet är jämnt och vid ingen paritet finns ingen paritetsbit med i överföringen.^{30 (s. 14)}

I sitt enklaste utförande sker kommunikation genom RS-232 protokollet med tre signaler. Transmit, Tx, för system 1 kopplas ihop med receive, Rx, på system 2 och transmit, Tx, på system 2 kopplas ihop med receive på system 1. Dessutom skickas signaljorden med för att säkerställa att sändare och mottagare känner av samma spänningsnivåer. Denna typ av överföring sker via en korsad kabel.^{30 (s. 15)}



Figur 2.13: Spänningsnivåer på bussen vid överföring av ett dataord.³¹

I sin enklaste implementeringen sker dataöverföring via RS-232 genom att data placeras i ett register, när sändaren är redo skiftas datan in i ett skiftregister, här adderas start-, stop och paritetsbit. Därefter skiftas datan ut på bussen via Tx pinnen. Då skiftregistret är tomt sätter RS-232-modulen flaggan "Transmission Data Register Emty" vilket medför att nästa dataord kan laddas i skiftregistret.^{30 (s. 16)} Data tas emot i mottagaränden genom att bitarna skiftas in i ett mottagarregister via Tx pinnen. Då flaggan "Reception Data Register Full" sätts är det möjligt att läsa in den överförda datan till ett valfritt register. Vid överläsning av data finns även kontrollregister som sätts beroende på värdet i paritetsbiten.^{30 (s. 16)}

3 VAL AV KOMPONENTER OCH UTVECKLINGSMILJÖ

3.1 Komponenter

3.1.1 Mikrokontroller

Initialt ställdes ett antal krav på mikrokontrollern. Dessa var att den skulle kunna drivas med enkel spänningsmatning, 0 till 5 Volt, använda 12-bitars A/D-omvandlare, innehålla stöd för CAN- och seriell kommunikation och finnas i både hålmonterad och ytmonterad version. Urvalet avgränsades ytterligare genom val av Microchip som tillverkare. Valet av Microchip baserades på att vi tidigare har erfarenhet av att använda mikrokontrollers från denna tillverkare.

Valet föll på dsPIC30F4013 från Microchip, vilket är en mikrokontroller med 16-bitars dataregister, 24-bitars adressregister och 2 kB arbetsminne. I övrigt uppfyller den alla de initiala kraven. Mikrokontrollern har stöd för både RS-232- och CAN-kommunikation. För mer information se avsnitt 2.8 : *Teknisk bakgrund: Kommunikation*. Den behöver dock kompletteras med transceiver-kretsar för att omvandla signalnivåer i enlighet med RS-232- och CAN-protokollet. Två kretsar, RS-232-modulen MAX232IN och CAN-modulen MCP2551 införskaffades för detta ändamål. Transceiver-kretsarna använder en så kallad charge pump för att skapa negativ spänning, detta eftersom övriga systemet är konstruerat med enkel spänningsmatning.

3.1.2 Ström och spänningsmätning

Vid val av strömmätningsgivare stod valet mellan halleffektsensor och strömshunt. För mer information se avsnitt 2.4 : *Teknisk bakgrund: Spännings- och strömmätningssmetoder*. Det som talar för en halleffektsensor är att den inte kräver någon förstärkarkrets utan direkt ger en för A/D-omvandlaren passande utsignal på 0 till 5 Volt. Dock är den på grund av magnetiseringsförluster behäftad med en större offset-spänning, vilket ger ett större procentuellt fel än strömshunten vid låga strömmar. Eftersom ett av de högre målen i projektet berör noggrannheten på strömmätningen valdes metoden med strömshunt. Valet av strömshunt föll på LA-100-100 från Canadian shunt. Shunten har en noggrannhet på $\pm 0,25 \%$ och ger en utspänning på 100 mV vid maxströmmen 100 A. För att kunna förstärka upp signalen om 100 mV till 5 V utan att tappa för mycket i precision krävs det en förstärkare med mycket bra upplösning. För mer information se avsnitt 2.7 : *Teknisk bakgrund: Icke ideala faktorer hos operationsförstärkare*. De val som lyftes fram bestod av spänning-till-frekvens omvandlare, Chopper-förstärkare och strömförstärkare.

Spänning-till-frekvens-omvandlaren valdes bort på grund av bristande noggrannhet, Chopper-förstärkaren valdes bort eftersom den krävde dubbel matning. För mer information om Chopper förstärkare se avsnitt 2.7 : *Teknisk bakgrund: Icke ideala faktorer hos operationsförstärkare*. Valet föll på strömförstärkaren LT6100 från Linear Technology. Förstärkaren har en noggrannhet på 0,5

% och tål insignaler upp till 48 Volt. Förstärkaren är konstruerad att förstärka spänningsfallet över en strömshunt som är inkopplad på den positiva spänningssidan i systemet. Nackdelen med en sådan konstruktion kan vara att det ställer höga krav på förstärkarens Common Mode Rejection Ratio (CMRR). För den aktuella förstärkaren var CMRR 120dB vilket anses vara mycket bra. Fördelen med att mäta strömmen på den positiva spänningssidan är att all ström inklusive eventuella läckageströmmar kommer med i mätningen.

Spänningsmätningen sker direkt från den positiva spänningssidan från strömshunten. För att anpassa spänningsvärdet till en nivå som A/D-omvandlaren kan hantera delas spänningen ner med hjälp av spänningsdelning. För mer information om nivåanpassning i ett digitalt elektriskt mätsystem se avsnitt 2.4 : *Teknisk bakgrund: Spännings- och strömmätningssmetoder*.

Spänningen som mäts mellan 0 och 30 volt delas ner till en nivå mellan 0 och 5 volt. Vid spänningsmätningen kommer mätvärdena alltid att befinna sig i den övre delen av intervallet, för att få bättre upplösning i mätningarna skulle därför detta intervall kunna anpassas för att bättre matcha de reella spänningsnivåerna. Vi ansåg dock att upplösningen var tillräckligt god då det maximala felet vid A/D omvandlingen kan beräknas till 0,025%.

3.1.3 A/D-omvandling

För att få god noggrannhet i A/D-omvandlingarna krävs en spänningsreferens med god noggrannhet. För detta ändamål övervägdes en spänningsreferens om 4,096 Volt. Fördelen med en sådan spänningsreferens är att det underlättar beräkningar i mjukvaran. A/D-värdet kommer då direkt att representera inspanningen i mV. Det som till slut gjorde att vi lämnade idén med en spänningsreferens om 4,096 Volt var att den valda strömförstärkaren ger en utspänning mellan 0 och 5 Volt. Varpå det slutgiltiga valet blev en spänningsregulator, MIC29150 från Micrel, på 5 Volt med ett spänningstolerans om 1 %. Med denna spänningsregulator regleras en spänning på 12 Volt via CAN-bussen ner till systemets matningsspänning på 5 Volt. Spänningen från regulatorn används således både som spänningsreferens och matning till systemet. För mer information om A/D-omvandling se avsnitt 2.6 : *Teknisk bakgrund: A/D-omvandling*.

3.1.4 Flashminne

Ett av kraven för att uppnå de högre målen i projektet är att systemet ska innehålla ett externt minne för möjligheten att ladda ned flera körcykler. För denna uppgift valdes ett EEPROM (Electrically Erasable Programmable Read-Only Memory). Kör-data som sparas undan är ström, spänning, laddningsgrad, hälsostatus och återstående körtid loggas. Denna data ska vara formaterad för att enkelt kunna infogas i ett kalkyldokument. Informationen kommer även att användas vid initiering av systemet då detta har befunnit sig i spänningslöst tillstånd. Det är tänkt att data ska skrivas till EEPROM en gång per 30 sekunder. Detta skulle kunna ske oftare om det finns behov, anledningen till att minska antal skrivningar till flashminnet är att detta slits. Det flashminnet vi valde är Microchip 24LC128 och har 16 kB minne. Flashminnet är konstruerat för att hålla en miljon läs/skriv cykler vid skrivning av en byte, vilket innebär att flashminnet är utslitet efter lite mindre än ett år i körtid (E_{timme}).

$$E_{time} = \frac{30 \cdot 1\,000\,000}{60 \cdot 60 \cdot 24} \approx 347 \text{ dagar}$$

Formel 3.1.

3.1.5 Övriga komponenter

I övrigt är systemet försett med avkopplingskondensatorer vid samtliga matningspunkter. Vid spänningsmatningen till systemet samt vid sammankopplingen mellan systemet och strömshunten sitter PTC-resistorer. Detta för att skydda systemet och begränsa då det av någon anledning skulle dra mer ström än vid normal drift.

3.2 Utvecklingsmiljö

3.2.1 MPLAB IDE

För att programmera mikroprocessorn används Microchips integrerade utvecklingsmiljö MPLAB. Programvaran är framtaget som ett hjälpmedel vid utveckling av applikationer till mikroprocessorer och digitala signalbehandlings processorer från Microchip.^{32 (s.19)} Programvaran innehåller projekthanterare, editor, länkare, debugger och execution engine. I projekthanteraren organiseras projektfilerna så att de enkelt kan skickas via kompilatorn till länkaren vid kompilering av applikationen. I Editorn matas den tänkta programkoden in. Länkaren länkar in den färdigkompileerade programvaran på rätt plats i mikroprocessorns minne vid programmering. Debuggern innehåller verktyg för att felsöka koden, den är direkt associerad med editorn för att härleda funktionen hos systemet tillbaka till koden. Execution engines simulerar hårdvaran, vilket gör att applikationen går att testa även då hårdvaran inte är färdigutvecklad.^{32 (s. 30)} MPLAB går att anpassa till ett stort antal mikroprocessorer genom val av kompilator. Vid utveckling av applikationer till dsPIC30F4013 rekommenderar Microchip, MPLAB C Compiler för dsPIC30F-serien av mikrokontrollers (MPLAB C30).³³ MPLAB tar applikationer skrivna i assembler, C eller basic och kompilerar denna till exekverbar kod som sedan kan programmeras till mikroprocessorn. MPLAB är kompatibelt med ett antal programmerare, däribland PICKit3 från Microchip.³⁴ I projektet används en gratisversion av kompilatorn MPLAB C30 vilket innebär att en del av den optimering av koden som sker i betalversionen inte finns tillgänglig.

3.2.2 Programmerare

PICKit3 är en programmerare/debugger som kontrolleras med hjälp av programvaran MPLAB IDE. Den är framtagen speciellt för mjukvaru- och hårdvarutveckling tillsammans med Microchip PIC mikroprocessorer och dsPIC signalprocessorer.^{35 (s. 13)} För programmering i samband med produktion finns det bättre programmerare att tillgå. PICKit3 fungerar bra med alla mikroprocessorer som stöder In Circuit Serial Programming (ICSP). Detta är ett programmeringssätt som tillåter mikroprocessorn att vara monterad i sin elektriska omgivning vid programmering.^{35(s. 13)} Genom PICKit3 kopplas målsystemet ihop med PC genom ett USB-interface. Tillsammans med MPLAB ger PICKit3 tillgång till alla tillgängliga egenskaper hos målsystemet. Bland annat kan mikroprocessorns register kontrolleras och modifieras från MPLAB IDE. Målsystemet kan även felsökas i realtid genom den inbyggda debug-funktionen.^{35(s. 15)}

4 FRAMTAGNING AV ELEKTRISKT SCHEMA

Följande avsnitt visar hur batteriövervakningssystemets olika komponenter är kopplade. Varje delkomponent redogörs var för sig, med avseende på förklaring av funktion, pin-konfiguration, dimensionering och koppling till andra komponenter. För en komplett bild av det elektriska schemat hänvisas till bilaga *Elektriskt schema*.

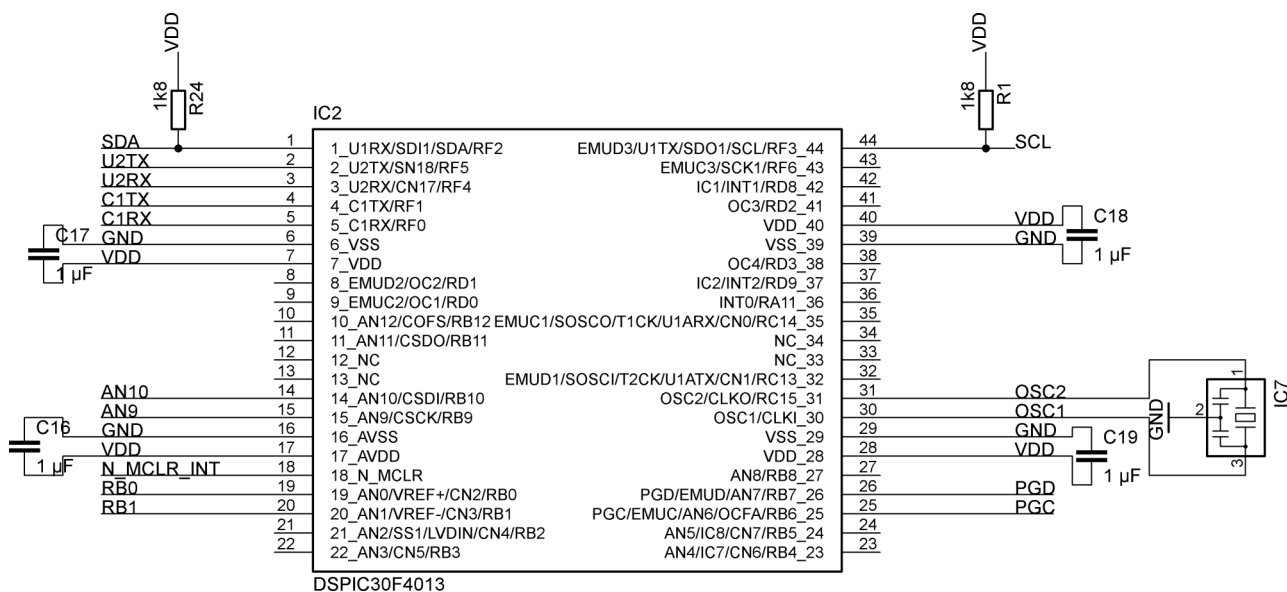
4.1 Mikrokontroller, Microchip dsPIC30F4013

Mikrokontrollern är kärnan i batteriövervakningssystemet och det är till denna alla andra kretsar kopplas. Mikrokontrollern har fyra par av pinnar för spänningsmatning (VDD och GND). Dessa måste separat kopplas av med kondensatorer. Dessutom är det viktigt att kondensatorerna sitter fysiskt nära mikrokontrollern så att impedansen förblir låg vid höga frekvenser. Avkopplingskondensatorerna är keramiska och har ett värde av 1 μF .

En keramisk resonator är inkopplad till mikrokontrollern via OSC1 och OSC2. Denna oscillerar med en frekvens om 10 MHz. Anledningen att denna används är att frekvenstoleransen är bättre för en extern oscillator ($\pm 0,5 \%$) än mikrokontrollerns interna ($\pm 2 \%$).

Till I²C-bussen (Inter-Integrated Circuit) är två stycken pull-up-resistorer inkopplade, en för den seriella dataledningen och en för den seriella klockledningen. Dessa behövs eftersom I²C-bussen använder sig av principen öppen kollektor och varje nod kan således vara recessiv eller dra bussen låg.

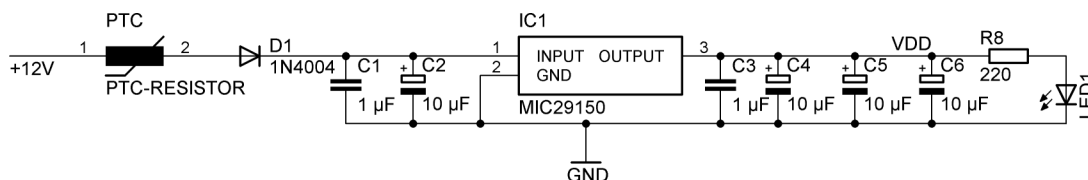
I figuren nedan är det markerat vilka av mikrokontrollerns pinnar som används samt vart dessa går. SDA och SCL är den seriella data respektive seriella klockan för I²C-bussen till flashminnet. U2TX samt U2RX är pinnar för sändning respektive mottagning till nivåomvandlaren för RS-232. AN10 är ingång för analog till digital-omvandling av spänningen från antivikningsfiltret vid strömförstärkaren. AN9 är ingång för spänningsmätning från spänningsdelningen och antivikningsfiltret. N_MCLR_INT är en intern reset-signal för mikrokontrollern från reset-nätet. RB0 och RB1 är pinnar som kan användas till felsökning eller senare påtänkta ändamål. PGD och PGC är pinnar för data respektive klocka vid programmering.



Figur 4.1: Mikrokontrollern, avkopplingskondensatorer, pull-up-resistorer och keramisk resonator.

4.2 Spänningsregulator, Micrel MIC29150

Spänningsregulatorns uppgift är att leverera stabil och exakt spänning till batteriövervakningssystemets alla delkomponenter. En PTC-resistor används för att skydda systemet vid en eventuell kortslutning. PTC-resistorn har en nominell hållström om 250 mA vilket således blir den högsta ström som regulatorn kan lämna. Efter denna sitter en likriktardiod som har till uppgift att skydda systemet vid inkoppling av matningsspänningen i fel riktning. För övrigt finns avkopplingskondensatorer, enligt rekommendationer från spänningsregulatorns datablad, samt en lysdiod för indikation av utspänning. Samtliga kondensatorer klarar en spänning om 35 Volt vilket således blir den maximala inspänningen för systemet, även om spänningsregulatorn sig klarar upp till 60 Volt.



Figur 4.2: PTC-resistor, likriktardiod, avkopplingskondensatorer, spänningsregulator, resistor och lysdiod.

4.3 Spänningsdelning

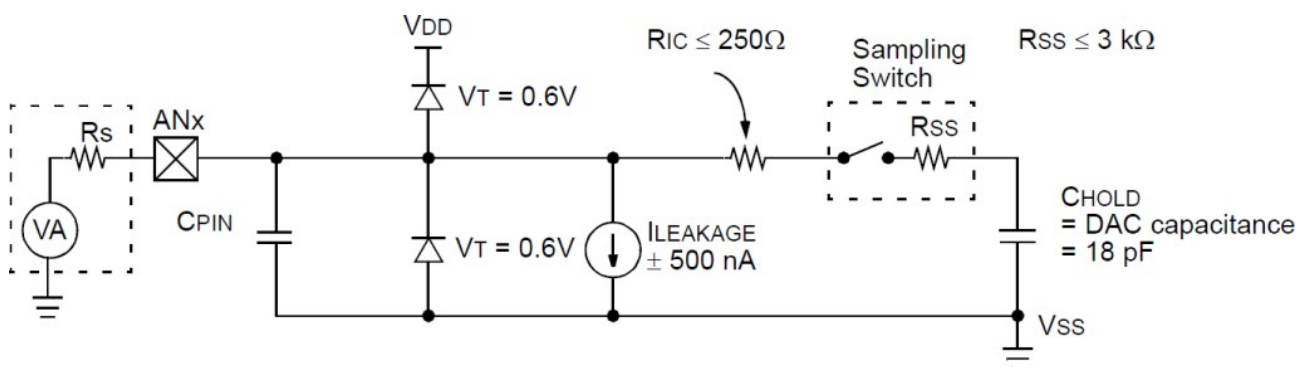
Spänningsdelningnätet är en del av batteriövervakningssystemets mätsystem. Därför krävs några av de delar hos ett mätsystem som presenteras i avsnitt 2.4 : *Teknisk bakgrund: Spännings- och strömmättningsmetoder*. De delar som används här är *ingångsskydd*, *nivåanpassning* och *filtrering*.

Spänningsdelningen är mätsystemets *nivåanpassning* och är nödvändig för att kunna mäta spänning i intervallet 0 till 30 Volt med en spänningsreferens om 5 Volt. Den spänningsdelning som är

nödvändig presenteras av sambandet nedan:

$$\frac{U_{REF}}{U_{MAX}} = \frac{5 \text{ Volt}}{30 \text{ Volt}} = \frac{1}{6} \Rightarrow \frac{R_2}{R_2 + R_3} = \frac{1}{6} \quad \text{Formel 4.1.}$$

Ingångsskyddet i spänningsmätningen hos mätsystemet utgörs av resistor R_3 tillsammans med de inbyggda skyddsdiодerna i mikrokontrollern. Resistansvärdena är dimensionerade så att den maximala strömmen in till, eller ut från, en ingång på mikrokontrollern begränsas vid en eventuell överspänning. En sådan överspänning kan till exempel uppkomma vid en elektrostatisk urladdning eller en felaktig inkoppling till en spänningskälla med hög spänning. I mikrokontrollern finns interna antiparallella skyddsdiодer kopplade mot jord respektive matningsspänning, se figuren nedan. Dessa begränsar en spänning in till mikrokontrollern till intervallet $-0,6 \rightarrow 5,6$ Volt förutsatt att strömmen genom dessa inte är över 25 mA och skyddar således mikrokontrollern från permanent skada.



Figur 4.3: Principschema över en ingång till mikrokontrollerns analog till digital-omvandlare.^{36 (s. 136)}

Den resistor som kan begränsa inströmmen vid en överspänning är R_3 (se figuren nedan). Dock gäller det att inte endast strömmen sätter gränsen för den maximala inspänningen till systemet. Även effekt- och spänningstoleransen hos resistorn kommer att sätta en gräns. Dimensioneringen gjordes med en grund att mätsystemet skall klara en inspänning om cirka 100 Volt utan att ta skada. Effekttoleransen hos resistorn ger då värdet hos R_3 enligt sambandet:

$$R_2 \geq \frac{U^2}{P} = \frac{(100 \text{ Volt})^2}{0,25 \text{ W}} = 40 \text{ k}\Omega, \left\{ \frac{R_2}{R_2 + R_3} = \frac{1}{6}, R_2 = 10 \text{ k}\Omega \right\} \Rightarrow \quad \text{Formel 4.2.}$$

$$\Rightarrow R_3 = 50 \text{ k}\Omega$$

För att anpassa resistorvärdet mot vad som finns tillgängligt att köpa samtidigt som förhållandet i spänningsdelningen upprätthålls valdes R_3 till 50 kΩ.

Mikrokontrollerns analog till digital-omvandlare har en läckström, kallad bias-ström (noterat som $I_{leakage}$ i figuren ovan). En enkel analys av vad denna läckströmmen får för inverkan hos mätvärdet visar:

$$U_{M\ddot{A}T} = (U_{IN} - R_3 I_{LEAKAGE}) \cdot \frac{R_2}{R_2 + R_3} \Rightarrow U_{M\ddot{A}T-FEL} = |R_3 I_{LEAKAGE}| \approx$$

Formel 4.3.

$$\approx 50 \text{ k}\Omega \cdot 0,61 \mu\text{A} \approx 30,5 \text{ mV}$$

I sambandet ovan är $U_{M\ddot{A}T}$ den spänning som systemet mäter, U_{IN} är inspänningen till mikrokontrollern samt $U_{M\ddot{A}T-FEL}$ mätfelet. Mätfelet förutsätter att källresistansen är noll samt antar värsta fallet för läckströmmen. Beloppet hos mätfelet anses vara tillräckligt bra för batteriövervakningssystemet.

Filtrering är den sista delen som erfordras av mätsystemet. Parallellt med spänningsdelningens utspänningsnod sitter en kondensator. Denna är till för att skapa ett första gradens lågpasfilter som används som ett antivikningsfilter hos systemet. Se avsnitt 5.1.6 : *Konfiguration av elektriska kretsar: Konfiguration av räknare* för information om antivikningsfiltrets gränsfrekvens.

Den viloström I_{U-Q} som går genom spänningsdelningsnätet kan beräknas med sambandet:

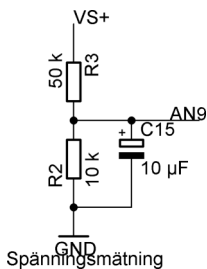
$$I_{U-Q} = \frac{U_{NOM}}{R_{IN}} + I_{LEAKAGE} = \frac{U_{NOM}}{R_2 + R_3} + I_{LEAKAGE} =$$

$$= \frac{22,2 \text{ Volt}}{10 \text{ k}\Omega + 50 \text{ k}\Omega} + 0,61 \mu\text{A} \approx 0,37 \text{ mA}$$

Formel 4.4.

I sambandet ovan är U_{NOM} den nominella batterispänningen i systemet. Vilostrommen visade sig senare under arbetet med test av systemet att vara icke försumbar. Mer om detta kan läsas under avsnitt 4.4 : *Framtagning av elektriskt schema: Strömförstärkare, Linear Technology LT6100* nedan.

Spänningsmätningen hos systemet kopplas in till strömshunten via VS+, se figuren nedan.



Figur 4.4: Resistorer och kondensator bildar spänningsdelning och antivikningsfilter.

4.4 Strömförstärkare, Linear Technology LT6100

Likt mätsystemet för spänningsmätning är strömmätningen en del av batteriövervakningssystemets mätsystem. Således innehåller också detta delarna *ingångsskydd*, *nivåanpassning* och *filtrering*. Skillnaden ligger här i komponenterna som utför dessa uppgifter.

Som *ingångsskydd* till strömmätningen används strömförstärkaren LT6100 från Linear Technology. Denna klarar en maximal inspänning om 48 Volt, vilket anses vara tillräckligt för systemet.

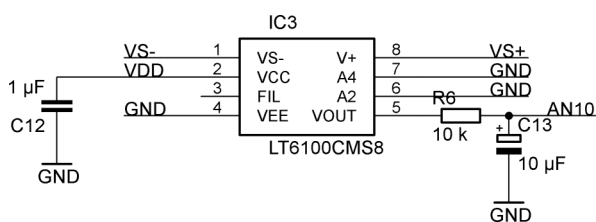
Nivåanpassningen i mätsystemet sköts också av strömförstärkaren. Inspänningen från strömshunten är $0 \rightarrow 100 \text{ mV}$ samt referensspänningen U_{REF} är 5 Volt vilket ger konfigurationen av strömförstärkarens förstärkning till 50 gånger. Detta väljs genom att ansluta ben 6 och 7 till jord.

Filtreringen i mätsystemet sker med hjälp av ett första gradens låpassfilter. Se avsnitt 5.1.6 : *Konfiguration av elektriska kretsar: Konfiguration av räknare* för information om antivikningsfiltrets gränsfrekvens.

Strömmätningen hos systemet kopplas in till båda sidorna av strömshunten, $VS+$ och $VS-$, se figuren nedan. Under utvecklingen av systemet var det tänkt att koppla dessa via PTC-resistorer till strömshunten. Detta för att ge ett skydd mot kortslutning vid ledningen till kretskortet eftersom strömshunten är kopplad till den positiva matningssidan och avsäkrad med en 40 A säkring. Dock försumrades den vilostrom I_{U-Q} som går genom spänningsdelningsnätet. Eftersom en PTC-resistor har en nominell resistans R_N vid rumstemperatur, kommer vilostrommen till spänningsdelningsnätet att orsaka ett spänningsfall U_{PTC} över PTC-resistorn enligt:

$$U_{PTC} = I_{U-Q} \cdot R_N = 0,37 \text{ mA} \cdot 5,6 \Omega \approx 2,1 \text{ mV} \quad \text{Formel 4.5.}$$

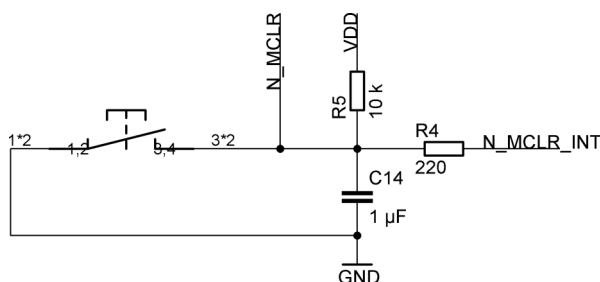
Strömshuntens givarkonstant om 1 mV/A motsvarar get att detta motsvarar ett mätfel för strömmätningen om 2,1 A, vilket för detta system är oacceptabelt. Som en tillfällig lösning gjordes därför valet att koppla förbi PTC-resistorn mellan $VS+$ och strömshunten och därmed eliminera detta spänningsfall. I en framtida version av modulen skulle detta problem enklast lösas genom att spännings- och strömmätningen använder separata PTC-resistorer.



Figur 4.5: Avkopplingskondensator, strömförstärkare samt resistor och kondensator utgör antivikningsfilter.

4.5 Reset-nät

Reset-nätets uppgift är att kunna starta om mikrokontrollern. Detta krävs vid programmering och görs av programmeraren via N_MCLR , se figuren nedan. När N_MCLR kopplas mot jord stannar exekveringen i mikroprocessorn, för att sedan starta om i programmets början när N_MCLR kopplas fri. N_MCLR kopplas till matningsspänning via en pull-up-resistor. Även en tryckknapp finns för att kunna starta om mikrokontrollern vid behov. En kondensator till jord skyddar mot oavsiktlig omstart vid störningar. Mot mikrokontrollern finns en resistor från reset-nätet för att skydda mot skadliga strömmar.



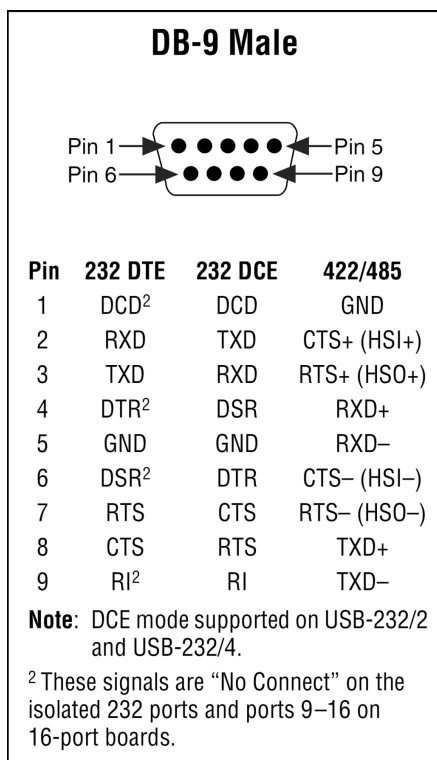
Figur 4.6: Resetknapp, pull-up-resistor, avkopplingskondensator och skyddsresistor.

4.6 Nivåomvandlare för RS-232-kommunikation, Texas Instruments MAX232

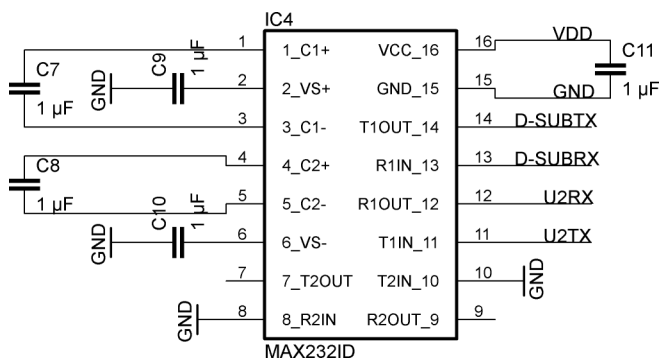
Nivåomvandlarens uppgift är att konvertera mikrokontrollerns signalnivåer om $0 \rightarrow 5$ Volt till RS-232-standardens spänningsnivåer om $7 \rightarrow -7$ Volt. För att göra detta med en enkel matningsspänning om 5 Volt använder kretsen en intern spänningsomvandlare av typen laddningspump (eng. charge pump). Fördelen med denna typ är att den endast kräver kondensatorer för att omvandla spänningen. Dimensioneringen av kondensatorerna följer databladets rekommendationer.

Nivåomvandlaren har möjlighet att omvandla två stycken RS-232-kanaler samtidigt. Eftersom endast en RS-232-kanal används, kopplas den andra kanalens ingångar för RX respektive TX, till jord. D-SUBTX kopplas till port 2 och D-SUBRX kopplas till port 3 på DE-9-kontaktdonet vidare till en seriell port på en PC, se figur nedan. På kontaktdonet kopplas även port 5 till jord. U2RX och U2TX är signaler för mottagning respektive sändning av data till och från mikrokontrollerns UART-modul.

Batteriövervakningsmodulen använder sig av en kontakt av typ DE-9-hona för RS-232-kommunikation. Kopplingen till serieporten på en PC kan då göras med okorsat DE-9-hona till DE-9-hane kablage.



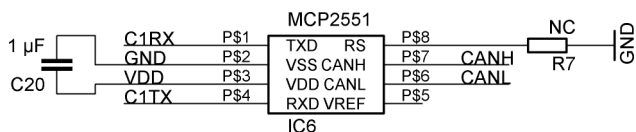
Figur 4.7: Pin-konfiguration för en seriell port med DE-9 (DB-9) kontaktdon.³⁷ DTE står för Data Terminal Equipment och är vanligtvis en terminal och DCE står för Data Communication Equipment och är vanligtvis kommunikationsutrustningen.



Figur 4.8: Nivåomvandlare för RS-232 med kondensatorer för inbyggd charge-pump och avkopplingskondensatorer.

4.7 Nivåomvandlare för CAN-kommunikation, Microchip MCP2551

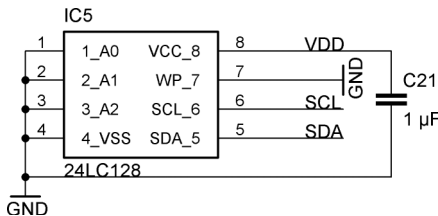
Nivåomvandlaren behövs för att omvandla mikrokontrollerns signalnivåer om 0 → 5 V från portarna C1RX och C1TX till en differentiell signal som kan kopplas till en CAN-buss enligt CAN-standard. CANH och CANL kopplas till CAN-bussens CAN-High respektive CAN-Low.



Figur 4.9: Nivåomvandlare för CAN-kommunikation med avkopplingskondensator och plats för att koppla in en resistor.

4.8 Flashminne, Microchip 24LC128

Flashminnet används av batteriövervakningssystemet för att kunna spara inställningar och logga data. Flashminnet är av typ seriell EEPROM (Electrically Erasable Programmable Read-Only Memory) och kommunicerar till mikrokontrollern via I²C-bussen. SCL är kopplat till I²C-bussens seriella klocksignal och SDA är kopplat till I²C-bussens seriella datasignal.



Figur 4.10: Flashminne och avkopplingskondensator.

4.9 Headers

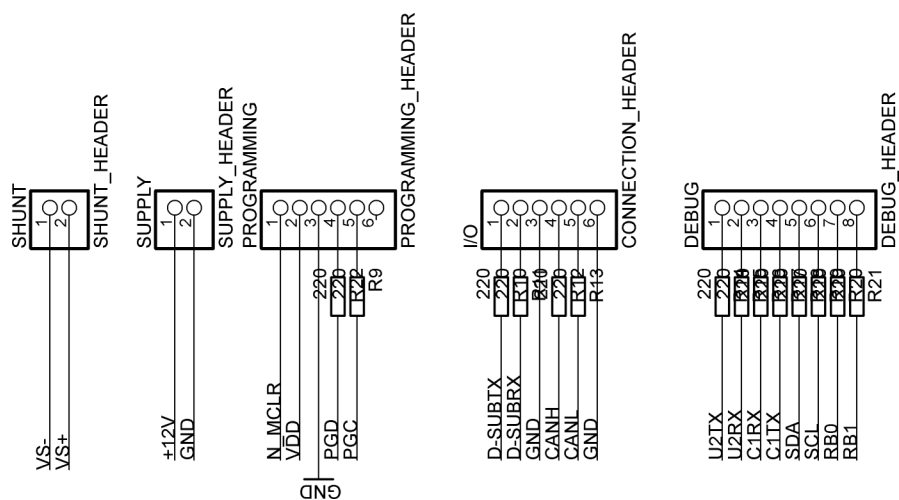
Headers är kontaktdon som stiftlistor kan anslutas till. De valda kontaktdonen använder en delning om 2,54 mm. Dessa används för att kunna ansluta batteriövervakningssystemet till andra komponenter. Fördelen med att inte löda kablar direkt ned i kretskortet är att det blir enklare att byta kretskortet i modulen samt att arbetet under utvecklingen blir smidigare då det går att koppla in olika utrustning.

Shunt header är anslutningen för spännings- och strömmätningen från strömshunten. VS+ skall kopplas till den sidan på strömshunten som är närmast batteriet och därmed har högst potential vid en urladdningström. VS- skall kopplas till den andra sidan strömshunten. *Supply header* används för spänningsmatning. *Programming header* används för att kunna programmera om mikrokontrollern. *Connection header* används för all extern kommunikation. Dessa är RS-232- och CAN-kommunikation. *Debug header* är tänkt att kunna användas främst under utveckling av modulen för att ha en möjlighet att mäta all intern kommunikation mellan mikrokontrollern och dess periferierheter. Till den interna kommunikationen hör U2TX och U2RX som är signaler mellan UART-modulen på mikrokontrollern samt nivåomvandlaren för RS-232-kommunikation för sändning respektive mottagning av data. C1RX och C1TX är signaler mellan CAN-modulen på mikrokontrollern till nivåomvandlaren för CAN-kommunikation för mottagning respektive sändning av data. SDA och SCL tillhör I²C-bussen som används för kommunikation mellan mikrokontrollern och flashminnet. RB0 och RB1 finns tillgängliga för att kunna användas vid felsökning, till exempel för att verifiera att en funktion körs med en viss frekvens, eller senare påtänkta ändamål.

Mellan alla anslutningar och modulens kretsar finns skyddsresistorer inkopplade. Dessa är dimensionerade så att en kortslutning mellan matningsspänningen om 5 Volt och jord inte överskrider strömmen 25 mA, vilket är den maximala strömmen mikrokontrollern klarar in till, eller ut från, en port utan att ta skada.

$$I_{MAX} = \frac{U_{MATNING}}{R_{SKYDD}} = \frac{5 \text{ Volt}}{220 \Omega} \approx 22,7 \text{ mA}$$

Formel 4.6.



Figur 4.11: Header-anslutningar för shuntspänning, spänningsmatning, programmering, kommunikation och felsökning.

5 KONFIGURATION AV ELEKTRISKA KRETSAR

För att ett elektriskt system ska fungera krävs att vissa av dess delkomponenter konfigureras för önskad funktion. I det här systemet för batteriövervakning finns sex stycken elektriska kretsar. Nedan redogörs för var och en av dem hur de är konfigurerade.

5.1 Mikrokontroller, Microchip dsPIC30F4013

Mikrokontrollern är kärnan i systemet och denna kräver mycket konfiguration för att önskade funktioner ska kunna användas. Mikrokontrollern består av ett antal olika moduler som var och en måste konfigureras för sig. För detaljerad information om konfigurationen hänvisas till källkoden som finns i bilagan samt Microchips manual för dsPIC30F-familjen³⁸ och databladet för mikrokontrollern dsPIC30F4013³⁹.

5.1.1 Konfiguration av klockfrekvensen

Genom så kallade konfigurationsbitar väljs oscillatorns mod till XT. Denna moden möjliggör att användandet av en extern kristall eller resonator mellan port OSC1 och OSC2. Oscillatorfrekvensen (f_{osc}) blir i denna mod samma som resonatorns frekvens, dvs. 10 MHz, vilket resulterar i att instruktionsfrekvensen (f_{cy}) blir 2.5 MHz. Valet av oscillatorfrekvens är gjord för en kompromiss mellan prestanda och strömförbrukning. Initialt under utveckling av modulen användes en oscillatorfrekvens om 40 MHz vilket resulterade i en strömförbrukning om cirka 86 mA. Detta är en relativt hög strömförbrukning som vid en linjär spänningsreglering till modulen från 12 V till 5 V resulterar i en effektförbrukning om cirka 1 W. Eftersom det visade sig att mikrokontrollerns beräkningsprestanda räckte till med stor marginal valdes det att sänka oscillatorfrekvensen till 10 MHz. Strömförbrukningen kunde genom denna ändring sänkas till cirka 25 mA för hela systemet, vilket anses mer rimligt. Effektförbrukningen minskades därigenom från cirka 1 W till 0,3 W. Värt att notera är att mikrokontrollern använder sig av en modifierad Harvard-arkitektur vilket innebär att majoriteten av alla instruktioner exekveras under en klockcykel.

```
/* Select XT oscillator mode => Fosc = 10 MHz => Fcy = 2.5 MHz. */  
_FOSC(XT)
```

Kod 5.1: Konfiguration av klockfrekvens.

5.1.2 Konfiguration av portar

På modulen finns två stycken portar, RB0 samt RB1, som finns tillgängliga för att kunna användas vid felsökning, eller senare påtänkta ändamål. Konfigurationen av dessa innefattar att välja dem som utgångar.

```

/* * * Port Configuration * * */

/* Set RB0 as output. */
TRISBbits.TRISB0 = OUTPUT;
/* Set RB1 as output. */
TRISBbits.TRISB1 = OUTPUT;

```

Kod 5.2: Konfiguration av portar.

5.1.3 Konfiguration av UART-modulen

UART-modulen (Universal Asynchronous Receiver Transmitter) används för RS-232-kommunikation mellan systemet och en dator. För modulen behöver först bithastigheten (eng. baud rate) ställas in. Detta görs enligt sambandet presenterat av formeln nedan, där U2BRG är registervärdet som konfigureras:

$$U2BRG = \frac{f_{CY}}{16 \cdot \text{Baud rate}} - 1 = \frac{2,5\text{MHz}}{16 \cdot 38,4\text{kb/s}} - 1 \approx 3,069... \Rightarrow \text{Formel 5.1.}$$

$$\Rightarrow U2BRG = 3$$

Valet är gjort så att högsta möjliga baud rate som tillhör standardvärdena kan väljas samtidigt som värdet som ställs in i registret U2BRG inte skiljer sig för mycket från det framräknade. Enligt sambandet ovan är en baud rate om 38,4 kb/s vald. Med baud rate inställt kan modulen startas och sändning aktiveras. Protokollet för dataöverföringen använder sig av standardinställningarna, 8 databitar och 1 paritetsbit.

```

/* * * UART Configuration * * */

/* Set the baud rate. */
U2BRG = 3;
/* Enable the UART2 module. */
U2MODEbits.UARTEN = true;
/* Enable transmission. */
U2STABits.UTXEN = true;

```

Kod 5.3: Konfiguration av UART-modulen.

5.1.4 Konfiguration av analog till digital-omvandlaren

Mikrokontrollern har en intern 12-bitars analog till digitalomvandlare som används för att mäta spänning och ström (genom en spänning från strömförstärkaren). Som referens till analog till digital-omvandlaren används jord och positiv spänningsmatning om 5 V. Det första som behöver konfigureras är den interna modulens omvandlingsklocka. Frekvensen för denna skall väljas så att en klockperiod (T_{AD}) blir minst 333,33 ns lång. Eftersom samplingsfrekvensen ($2 \cdot 500$ Hz) i detta system är relativt låg i förhållande till den maximala samplingsfrekvens (200 kHz) som analog till digital-omvandlaren klarar. Genom att välja den långsammaste omvandlingsklockan med hjälp av

ADCS = 63 erhålls bäst prestanda av analog till digital-omvandlaren med avseende till störningar. Klockperioden blir då enligt formeln nedan:

$$T_{AD} = \frac{T_{CY} \cdot (ADCS + 1)}{2} = \frac{ADCS + 1}{2 \cdot f_{CY}} = \frac{63 + 1}{2 \cdot 2,5 \text{ MHz}} = 12,8 \mu s \quad \text{Formel 5.2.}$$

Detta uppfyller väl kravet på minsta klockperiod (T_{AD}). En minimal komplett analog till digital omvandlingscykel tar enligt databladet 15 klockperioder och därför blir den maximala möjliga samplingsfrekvensen, med vald omvandlingklocka, enligt nedan:

$$f_{S-MAX} = \frac{1}{15 \cdot T_{AD}} = \frac{1}{15 \cdot 12,8 \mu s} \approx 5,208 \text{ kHz} \quad \text{Formel 5.3.}$$

Den interna analog till digital-omvandlaren konfigureras för att automatiskt börja sampla signalen efter varje konvertering. Detta ger en så lång samplingstid som möjligt. Vidare väljs att samplingarna växelvis ska göras mellan MUXA och MUXB. För MUXA väljs kanal AN9 samt för MUXB väljs kanal AN10. På detta sätt kan sampling av två signaler göras effektivt med en A/D-omvandlare. För vartannat sampel genereras ett avbrott. Sist aktiveras A/D-omvandlaren.

```
/* * * ADC Configuration * * */

/* Set the A/D conversion clock. */
ADCON3 |= 63;
/* Start sampling after each conversion. */
ADCON1bits.ASAM = true;
/* Alternate the sampling between MUXA and MUXB. */
ADCON2bits.ALTS = true;
/* Set MUXA to An9 and MUXB to An10. */
ADCHS |= 0x0A09;
/* Generate an interrupt once every two samples. */
ADCON2 |= 0x0004;
/* Enable the A/D module. */
ADCON1bits.ADON = true;
```

Kod 5.4: Konfiguration av analog till digital-omvandlaren.

5.1.5 Konfiguration av I²C-modulen

Mikrokontrollerns I²C-modul (Inter-Integrated Circuit) används för att kommunicera med ett EEPROM. På grund av ett hårdvarufel i denna revision av mikrokontroller måste först en nolla skrivas till latch F2. Efter att detta är gjort väljs en bithastighet, dvs. baud rate, enligt nedan. Den seriella klockhastigheten (F_{SCL}) för bussen är vald till den lägsta möjliga om 100 kHz, enligt I²C-standardens.

$$I2CBRG = \frac{f_{CY}}{f_{SCL}} - \frac{f_{CY}}{1\,111\,111} - 1 = \frac{2,5 \text{ MHz}}{100 \text{ kHz}} - \frac{2,5 \text{ MHz}}{1\,111\,111} - 1 \approx 21,75 \Rightarrow I2CBRG = 22 \quad \text{Formel 5.4.}$$

Med inställningen av baud rate gjord aktiveras I²C-modulen.

```
/* * * I2C Configuration * * */  
  
/* Workaround built in error. */  
LATFbits.LATF2 = false;  
/* Set the baud rate. */  
I2CBRG = 22;  
/* Enable the I2C-module. */  
I2CCONbits.I2CEN = true;
```

Kod 5.5: Konfiguration av I²C-modulen.

5.1.6 Konfiguration av räknare

Mikrokontrollern har olika räknare som kan användas för diverse ändamål såsom tidtagning och avbrott. Timer 1 kan användas för att implementera en realtidsklocka. Med anledning av detta har timer 1 lämnats ledig för eventuella framtida behov. Timer 2 har valts att användas för att generera periodiska avbrott med en frekvens om 1 kHz. I avbrottsrutinen för räknaren startas analog till digital-omvandlingarna och på så sätt erhålls den önskade samplingsfrekvensen om 500 Hz per kanal.

Valet av samplingsfrekvens är baserat på att gränshfrekvensen ($f_{\text{spänning}, G}$ respektive $f_{\text{ström}, G}$) på systemets båda antivikningsfilter:

$$f_{\text{spänning}, G} = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 50\text{ k}\Omega // 10\text{ k}\Omega \cdot 10\text{ }\mu\text{F}} \approx 1,910\text{ Hz} \quad \text{Formel 5.5.}$$

$$f_{\text{ström}, G} = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 10\text{ k}\Omega \cdot 10\text{ }\mu\text{F}} \approx 1,592\text{ Hz} \quad \text{Formel 5.6.}$$

Samplingsteoremet säger att det då är nödvändigt att sampla med minst cirka 4 Hz för att kunna återskapa denna signalen. Eftersom systemet använder första gradens lågpasfilter som antivikningsfilter gäller det att frekvenskomponenter över 1,9 Hz respektive 1,6 Hz dämpas med 20 dB per dekad. Genom att använda en samplingsfrekvens om 500 Hz dämpas signaler vid halva samplingsfrekvensen enligt nedan:

$$\begin{aligned} A_{\text{spänning}, 250\text{ Hz}} &= \frac{1}{\sqrt{1 + (2\pi f RC)^2}} = \\ &= \frac{1}{\sqrt{1 + (2\pi \cdot 250\text{ Hz} \cdot 50\text{ k}\Omega // 10\text{ k}\Omega \cdot 10\text{ }\mu\text{F})^2}} \approx -42\text{ dB} \end{aligned} \quad \text{Formel 5.7.}$$

$$A_{ström, 250 Hz} = \frac{1}{\sqrt{1 + (2\pi f RC)^2}} =$$

Formel 5.8.

$$= \frac{1}{\sqrt{1 + (2\pi \cdot 250 Hz \cdot 10 k\Omega \cdot 10 \mu F)^2}} \approx -44 dB$$

Detta betyder att störningar vid 250 Hz kommer att dämpas med 42 dB respektive 44 dB. Eftersom systemet kommer att mäta likspänning och -ström anses detta tillräckligt.

För att välja med vilken frekvens som avbrotten ska genereras av timer 2 används följande formel för att konfigurera registret PR2:

$$PR2 = \frac{f_{CY}}{f_{TMR}} = \frac{2,5 MHz}{1 kHz} = 2500$$

Formel 5.9.

Med inställningen av periodregistret gjord kan timer 2 aktiveras.

```
/* * * Timer 2 Configuration * * */
/* Set timer 2 period. */
PR2 = 2500;
/* Turn on timer2. */
T2CONbits.TON = true;
```

Kod 5.6: Konfiguration av timer 2.

Timer 3 används för att starta en avbrottsrutin som ska göra periodiska beräkningar en gång per sekund. Dock går det inte att direkt välja en så hög inställning i periodregistret. Därför måste en så kallad nedskalare (eng. prescaler) användas som klockkälla för timer 3. En lämplig inställning för prescalern är att den dividerar instruktionscykelklockan med 64 och därav blir formeln för inställning av periodregistret (PR3) för timer 3:

$$PR3 = \frac{f_{CY} / presc.}{f_{TMR}} = \frac{f_{CY}}{presc. \cdot f_{TMR}} =$$

$$= \frac{2,5 MHz}{64 \cdot 1 Hz} = 39062,5 \Rightarrow PR3 = 39063$$

Formel 5.10.

Med inställningen av prescalern och periodregistret gjord kan timer 3 aktiveras.

```
/* * * Timer 3 Configuration * * */
/* Set timer 3 prescaler to 1:64. */
T3CONbits.TCKPS1 = true;
T3CONbits.TCKPS0 = false;
/* Set timer 3 period. */
PR3 = 39063;
/* Turn on timer 2. */
T3CONbits.TON = true;
```

Kod 5.7: Konfiguration av timer 3.

Timer 4 används för att mäta den tid som spenderas i avbrottsrutiner. Detta möjliggör en uppskattning av hur utnyttjandegraden av systemets beräkningskapacitet är. Genom att aktivera timer 4 i början av en avbrottsrutin och sedan deaktivera den i slutet kan exekveringstiden för avbrottsrutiner mätas. För timer 4 ställs prescalern till 1:64 så att tider något över en sekund kan mätas. Timer 4 aktiveras inte i initieringen utan detta sker i de avbrottsrutiner för vilka exekveringstiden skall mätas.

```
/* * * Timer 4 Configuration * * */

/* Set timer4 prescaler to 1:64. */
T4CONbits.TCKPS1 = true;
T4CONbits.TCKPS0 = false;
```

Kod 5.8: Konfiguration av timer 4.

Timer 5 används tillsammans med timer 4 för köra en avbrottsrutin en gång i sekunden där processorns utnyttjandegrad beräknas. Anledningen till att inte avbrottsrutinen för timer 3, som också körs en gång i sekunden, kan användas är att exekveringstiden för den avbrottsrutinen också önskas mätas. Likt timer 4 ställs prescalern för timer 5 till 1:64. Dessutom konfigureras periodregistret (PR5) så att avbrott sker en gång i sekunden:

$$PR5 = \frac{f_{CY} / presc.}{f_{TMR}} = \frac{f_{CY}}{presc. \cdot f_{TMR}} = \frac{2,5 MHz}{64 \cdot 1 Hz} =$$

Formel 5.11.

$$= 39062,5 \Rightarrow PR5 = 39063$$

Med inställningen av prescalern och periodregistret gjord kan timer 5 aktiveras.

```
/* * * Timer 5 Configuration * * */

/* Set timer 5 prescaler to 1:64. */
T5CONbits.TCKPS1 = true;
T5CONbits.TCKPS0 = false;
/* Set timer 5 period. */
PR5 = 39063;
/* Turn on timer 5. */
T5CONbits.TON = true;
```

Kod 5.9: Konfiguration av timer 5.

5.1.7 Konfiguration av CAN-modulen

Mikrokontrollerns CAN-modul (Controller Area Network) används för att skicka information från batteriövervakningsmodulen till andra delar av systemet. Den information som skickas är de parametrar som beräknas av modulen. För att informationen skall kunna skickas måste ett antal inställningar hos modulen konfigureras.

Det första som behöver konfigureras är CAN-modulens klockhastighet, det vill säga dess baud rate. För detta system önskas en baud rate för CAN-modulen om 250 kb/s. CAN-modulens baud rate kan

konfigureras dels med en prescaler och dels med hur många gånger instruktionsklockan ska delas med 2. Genom att välja antalet tidskvanta (T_Q) per bittid till $K = 10$ erhålls T_Q enligt nedan. Observera att K måste väljas inom intervallet 8 till 25.

$$T_Q = \frac{1 / \text{Baud rate}}{K} = \frac{1/250\text{kbit/s}}{10} = 0,4\mu\text{s} \quad \text{Formel 5.12.}$$

Givet tidskvantan T_Q kan sedan registervärdet BRP ställas in med hjälp av sambandet:

$$BRP = \frac{2T_Q}{T_{CY}} - 1 = 2T_Q F_{CY} - 1 = 2 \cdot 0,4\mu\text{s} \cdot 2,5\text{MHz} - 1 = 1 \quad \text{Formel 5.13.}$$

Efter att registervärdet BRP är inställt kan nu CAN-modulens operationsmod bytas från "Configuration mode" till "Normal Operation mode". Noden tillåts dock inte att byta operationsmod under alla förhållanden utan kräver bland annat att bussen skall vara ledig. För att försäkra sig om att CAN-modulen har bytt till "Normal Operation mode" är det sista som görs i konfigurationen således att vänta på transitering till denna mod.

```
/* * * CAN Configuration * * */

/* Set CAN baud rate. */
C1CFG1 |= 1;
/* Request normal operation mode. */
C1CTRL = 0x0800;
/* Wait for normal operation mode. */
while(C1CTRL & 0x00E0);
```

Kod 5.10: Konfiguration av CAN-modulen.

5.1.8 Konfiguration av avbrott

I detta systemet finns inget behov av att använda prioritet mellan avbrotten. Dessutom kan inte nästlade avbrott tillåtas om mätningen av avbrottsrutinernas exekveringstid skall fungera. Nästlade avbrott innebär att ett pågående avbrott kan avbrytas av ett annat avbrott med högre prioritet. På grund av detta deaktiveras först nästlade avbrott. Därefter aktiveras avbrott för timer 2, timer 3, timer 5 och analog till digital-omvandlaren. Aktiveringen av avbrott är det sista som görs i konfigurationen av mikrokontrollern. Detta sker sist för att inte annan konfiguration skall störas.

```
/* * * Interrupt Configuration * * */

/* Disable interrupt nesting. */
INTCON1bits.NSTDIS = true;
/* Enable Timer2 interrupt. */
IEC0bits.T2IE = true;
/* Enable Timer3 interrupt. */
IEC0bits.T3IE = true;
/* Enable Timer5 interrupt. */
IEC1bits.T5IE = true;
/* Enable A/D interrupt. */
IEC0bits.ADIE = true;
```

Kod 5.11: Konfiguration av avbrott.

5.2 Nivåomvandlare för RS-232-kommunikation, Texas Instruments MAX232

Nivåomvandlarens (eng. line transceiver) uppgift är att omvandla mikrokontrollerns signalnivåer om $0 \rightarrow 5\text{ V}$ till $7 \rightarrow -7\text{ V}$ och vice versa. Konfigurationen av modulen begränsas till att den andra sändaren och mottagaren kopplas till jord för att dessa ingångar inte ska lämnas flytande.

5.3 Strömförstärkare, Linear Technology LT6100

Strömförstärkarens (eng. current sense amplifier) uppgift är förstärka spänningen från strömshunten. Spänningen från strömshunten är proportionell mot strömmen med en faktor 1 mV/A . Den maximala tillåtna strömmen genom shunten är 100 A och således 100 mV . Strömförstärkaren konfigureras därför till förstärka shuntspänningen med 50 gånger så att spänningen till mikrokontrollerns analog till digital-omvandlare blir mellan 0 och 5 V . Konfigurationen görs genom att port 6 och 7 jordas.

5.4 Spänningsregulator, Micrel MIC29150

Spänningsregulatorn förser batteriövervakningsmodulen med stabil 5 V linjärt reglerat från övriga systemets 12 V -matning. Spänningsregulatorn fungerar också som spänningsreferens till mikrokontrollerns analog till digital-omvandlare. Ingen speciell konfiguration av denna krets behöver göras.

5.5 Flashminne, Microchip 24LC128

Flashminnet (Serial EEPROM) används av systemet för att spara inställningar och logga data. Detta är inkopplat på I²C-bussen till mikrokontrollern. Konfigurationen innefattar dels till att kretsens adress på I²C-bussen är konfigurerad till "000" genom att adressport $0 \rightarrow 3$ (port $1 \rightarrow 3$) är kopplade till jord. Dels är skrivskyddet permanent deaktiverat genom att port 7 är kopplad till jord.

5.6 Nivåomvandlare för CAN-kommunikation, Microchip MCP2551

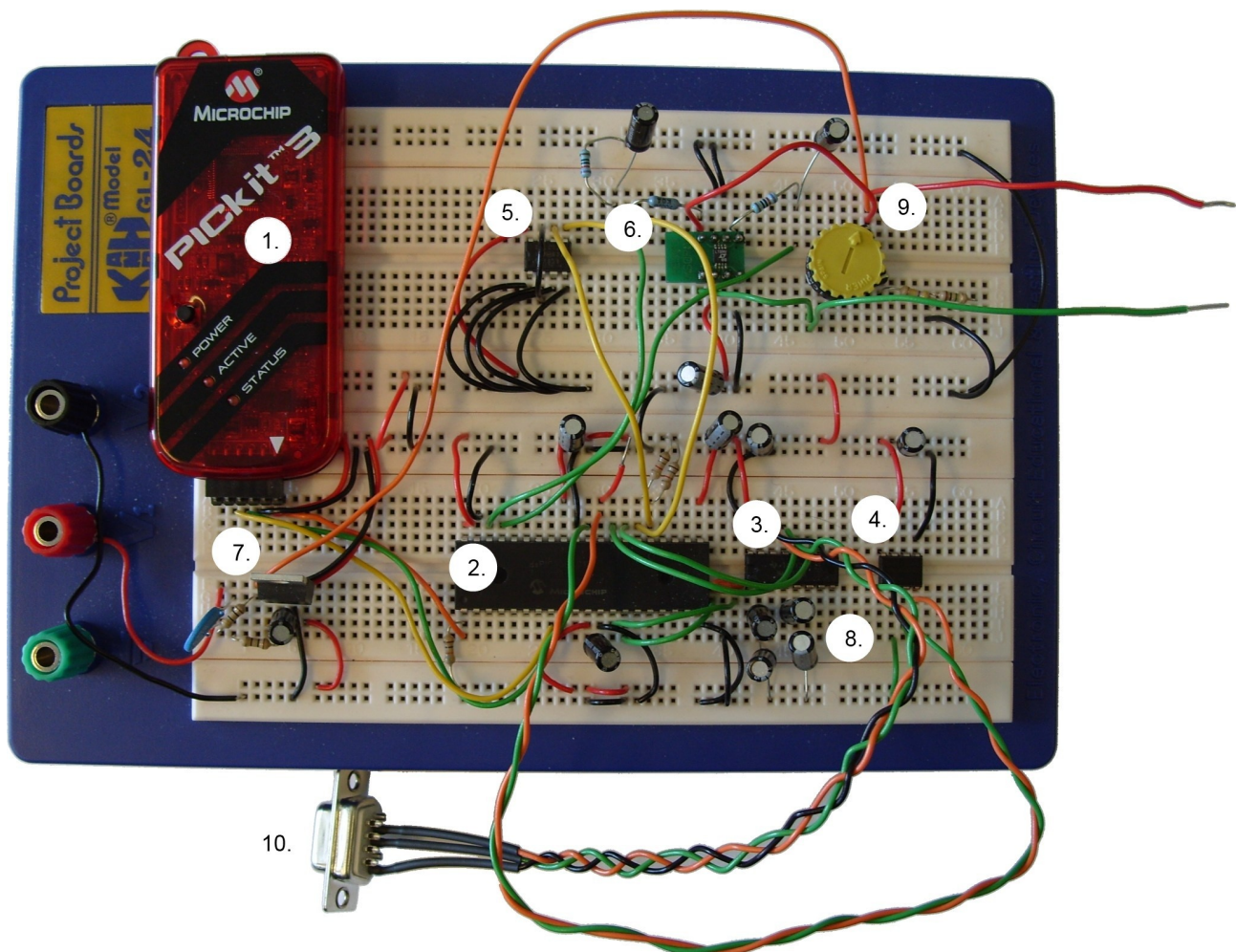
Nivåomvandlarens (CAN Transceiver) uppgift är att omvandla mikrokontrollerns signalnivåer om $0 \rightarrow 5\text{ V}$ till en differentiell signal bestående av CAN-High och CAN-Low. Ingen speciell konfiguration av denna krets behöver göras.

6 PROTOTYPUTVECKLING

Utveckling och test av batteriövervakningssystemets hårdvara och mjukvara har gjorts med hjälp av hålmonterade komponenter på ett kopplingsdäck. Fördelen med att använda ett kopplingsdäck vid hårdvaruutveckling är enkelheten att vid behov byta ut komponenter, samt att det ger en möjlighet att testa att komponenter har kopplats in korrekt innan en konstruktion av kretskort genomförs. I projektet har prototypen fungerat som målsystem vid mjukvaruutveckling eftersom det färdiga kretskortet anlände efter att majoriteten av mjukvaran färdigställts. All konfigurering av mikroprocessorn har genomförts och testats på detta system.

Insignalerna till systemet har simulerats, spänningen direkt via nätaggregat och strömmen via en potentiometer. Simulering av strömmen till mätsystemet består av att spänningsskillnaden på 0 – 100 mV över potentiometern motsvarar den spänning som uppstår över strömshunten då strömmen varierar mellan 0 och 100 Ampere. Med denna uppsättning har bland annat systemets A/D-omvandling och algoritm preliminärt testats.

På nästa sida presenteras en bild av komponenterna på kopplingsdäcket samt hur de är placerade.



Figur X.X: Bild över kopplingsdäcket som användes vid prototyputvecklingen.

1. Programmerare PICkit3.
2. Mikrokontroller dsPIC30F4013.
3. RS-232 tranceiver MAX232.
4. CAN tranceiver MCP2551.
5. EEPROM 24LC128.
6. Strömförstärkare LT6100 monterad på breakoutboard.
7. Spänningsregulator MIC29150.
8. Avkopplingskondensatorer.
9. Potentiometer för simulering av shuntspänning till strömförstärkaren.
10. D-sub 9 kontakt för RS-232 kommunikation med dator.

7 KRETSKORTSKONSTRUKTION

7.1 Val av komponenter

Vid framtagning av prototyp användes hålmonterade komponenter, på kretskortet används nästan uteslutet ytmonterade komponenter. Det första steget i konstruktionen blir därför att leta rätt på ytmonterade motsvarigheter till de hålmonterade. I komponenternas datablad finns en sammanställning av dess samtliga förpackningsversioner. I samtliga fall förutom CAN-transceiver användes dess ytmonterade motsvarighet. Anledningen till att vi valde en annan CAN-transceiver är att den har används på övriga noder på bilen. Eftersom komponenterna skall lödas för hand valdes för ytmonterade resistorer och keramiska kondensatorer den största tillgängliga förpackningen 1206. Samma tankesätt styrde även valet av förpackning på elektrolyt kondensatorerna. På kretskortet finns även tre hålmonterade komponenter, dessa är PTC-motstånd, likriktardiod och lysdiod. På kretskortet finns även anslutning för 12 Volt spänningsmatning, strömshunt, programmerare, RS-232-och CAN-kommunikation. Dessutom finns där en 8-polig debug-header där interna signaler mellan mikroprocessorn och transceiver kretsarna kan mätas. Till debug-headern är även två ben från processorn utdragna för framtida felsökning.

7.2 EAGLE

Framställningen av tillverkningsunderlag för kretskortstillverkning har gjorts i en gratisversion av CAD-programmet EAGLE (Easily Applicable Graphical Layout Editor). EAGLE är framtaget för att vara ett bra budgetalternativ till dyrare programvara för kretskortskonstruktion. Gratis versionen har vissa begränsningar. Storleken på kretskortet är begränsat till 100 * 80 mm, det finns endast möjlighet att använda två signallager och schemaritningen får endast göras på en sida i programvaran. Dessutom får programvaran ej användas i kommersiellt syfte.⁴⁰ För det aktuella projektet är dock dessa begränsningar ej till problem då storleken på den analoga konstruktionen är relativt liten.

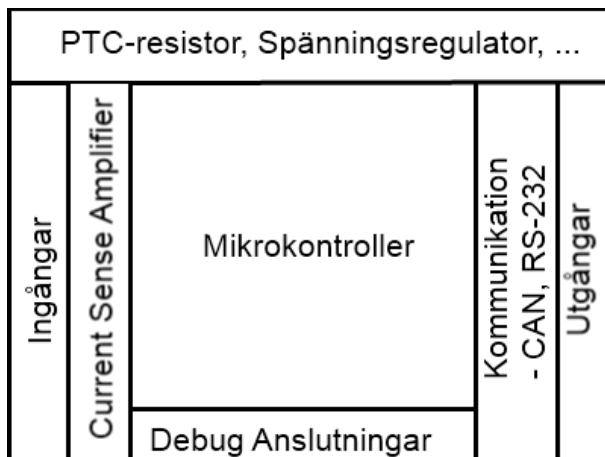
EAGLE består av de tre delarna schema verktyg, layout verktyg och biblioteks verktyg.⁴¹ I schema delen beskrivs hur kretsarna i den analoga konstruktionen ska sammankopplas. Då den sökta komponenten inte finns i standardbiblioteket finns det i EAGLE en editor för att rita egna komponenter både vad avser schemaritningen och layout ritningen. Dessa två versioner av komponenten länkas samman så att den komponent som infogas i schemaritningen automatiskt dyker upp i layout-delen. Komponenterna kan sedan arrangeras i bibliotek efter komponenttyp och tillverkare för framtida användning. I layout verktyget placeras komponenterna, jordplan, viahål och ledningsbanor ut. I layouten bestäms hur komponenterna kopplas in och monteras rent fysiskt. De kopplingar som gjorts i schema-delen avspeglas som hjälplinjer i layout-delen. I EAGLE finns en autoroute funktion, med hjälp av den kan programmet automatiskt koppla in alla komponenter efter att de placerats på önskad position. Auto-route funktionen ger ett elektriskt fungerande kretskort, dock finns det vissa aspekter den inte tar hänsyn till vad gäller stabiliteten på systemet.

7.3 Konstruktionsförfarande

Innan konstruktionsförfarandet inleds finns mikroprocessorns benkonfiguration klar. Det är alltså på förväg bestämt vilka ben som ska användas till A/D omvandlare, RS-232 kommunikation, CAN-kommunikation osv.

Konstruktionen av kretskortet börjar med att de komponenter som sedan tidigare inte finns i komponent biblioteket läggs till. Därefter konsulteras datablad för var och en av kretsarna så att dessa konfigureras och sammankopplas korrekt i schema-delen.

Innan arbetet i layout delen inleds görs en grov skiss över placeringen av komponenterna.



Figur 7 1: Grov skiss över komponentplacering på kretskort.

Vid utveckling av kretskorts layout för system med hög frekventa signaler finns det några saker som är värt att beakta. Dessa aspekter är egentligen inte avgörande för funktionen hos det aktuella systemet då dess signaler har relativt sett låg frekvens. Vi har ändå valt att designa layouten efter dessa då det kommer att hjälpa oss inför framtida kretskortskonstruktion.

Homogent jordplan: Försök att dra så få ledare som möjligt genom jordplanet. Där en sådan dragning inte går att undvika, minimera sträckan.

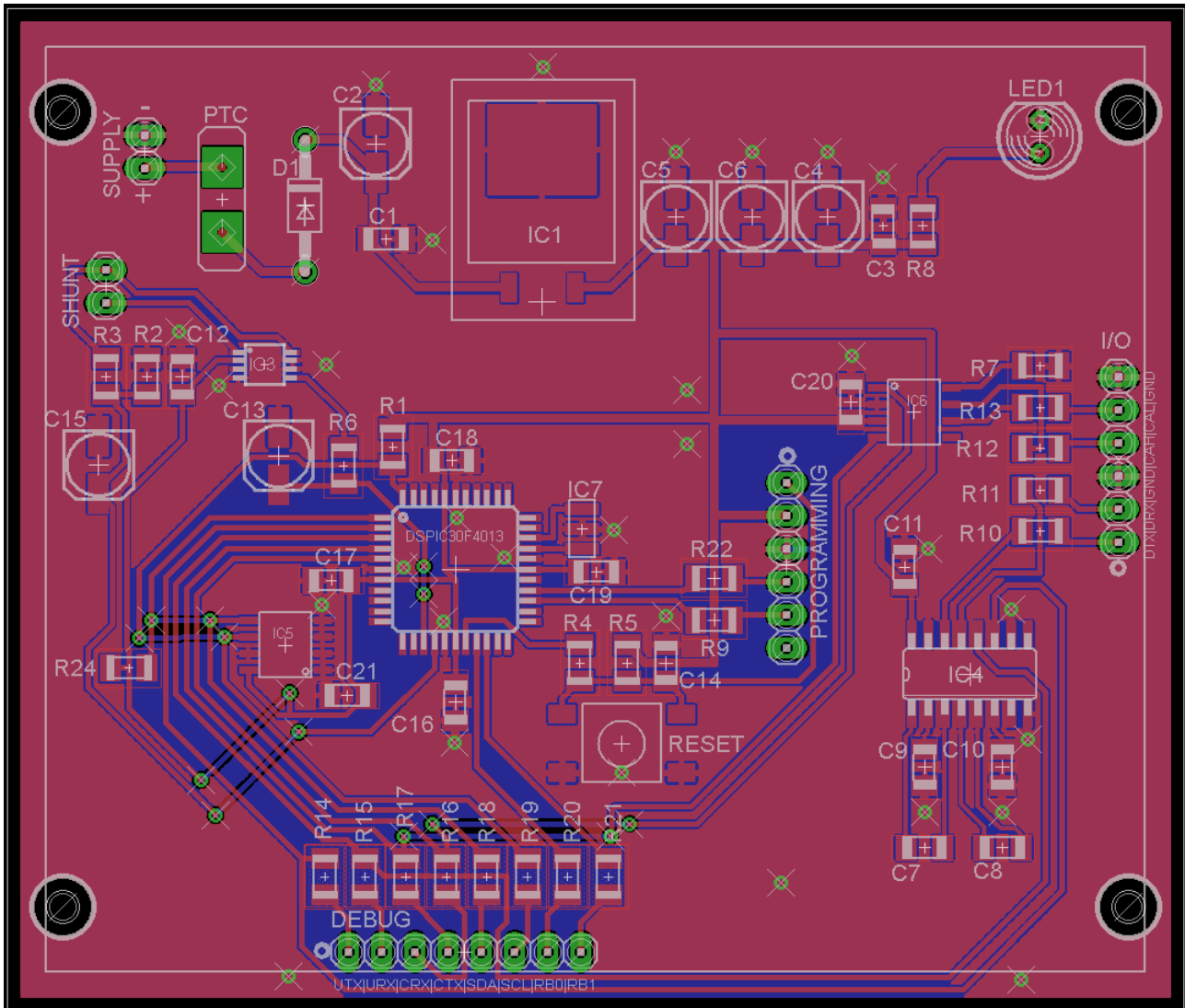
Dra ledningar i 45 grader: Anledningen till detta är att styrkan på E-fältet vid höga frekvenser och 90-gradiga ledningsböjar kan bli tillräckligt stort för att störa intilliggande ledningsbanor och komponenter.⁴²

Jordplan på övre lagret: Med två stycken jordplan, ett på undre planet och ett på det övre planet kortet inkapslat och mindre känsligt för störningar utifrån. Om kretskortet ska användas i miljöer med starka E- och B-fält kan viahål som binder ihop de två jordplanen även placeras runt kanten på kretskortet.

En snygg layout är en bra layout: Raka enkla ledningsbanor underlättar felsökning.

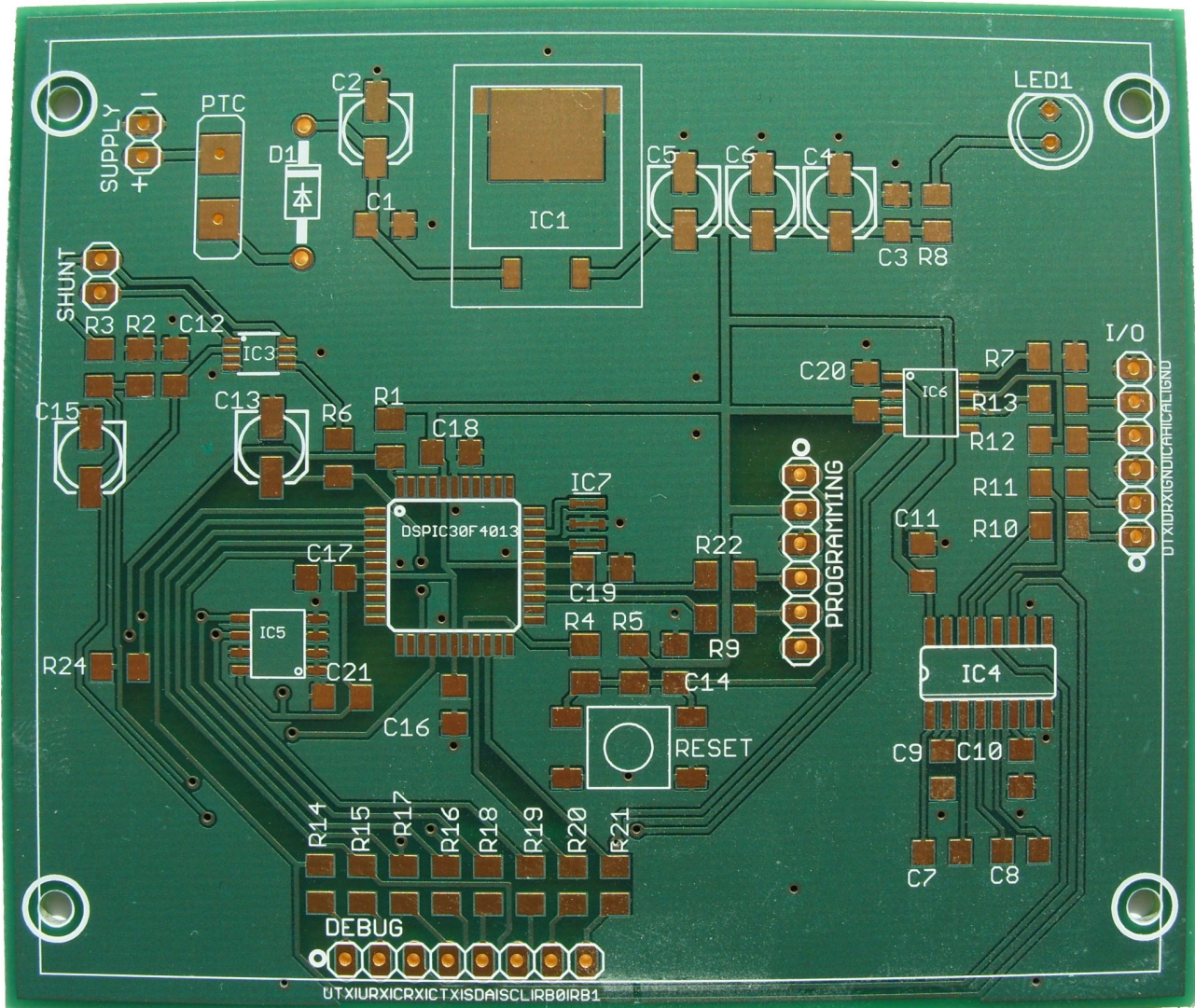
7.4 Resultat av kretskortskonstruktion

Nedan visas den resulterande kretskortslayouten. Utifrån denna layout togs tillverkningsunderlaget fram.



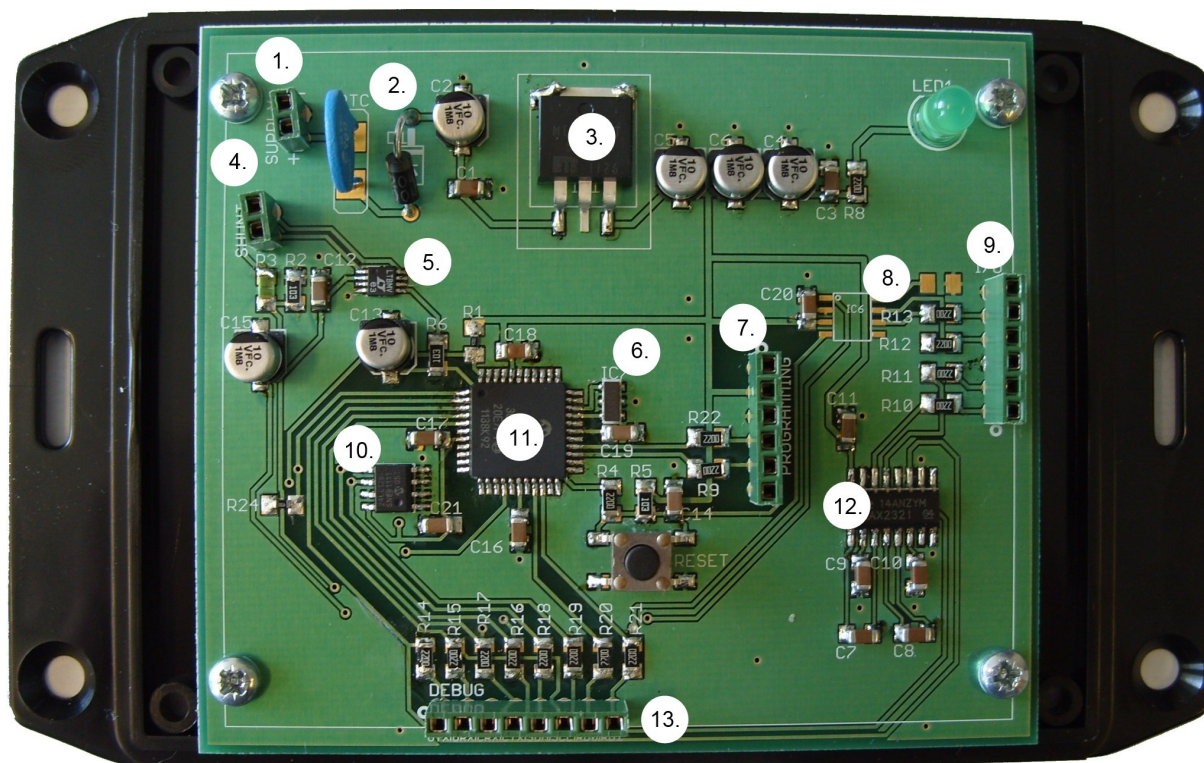
Figur 7 2: Resultatet av kretskorts design i EAGLE.

Nedan visas det tillverkade kretskortet innan montering av komponenter.



Figur 7 3: Kretskort före montering av komponenter.

Nedan visas kretskortet i sin slutgiltiga version monterat i bottenplattan till modul-lådan.



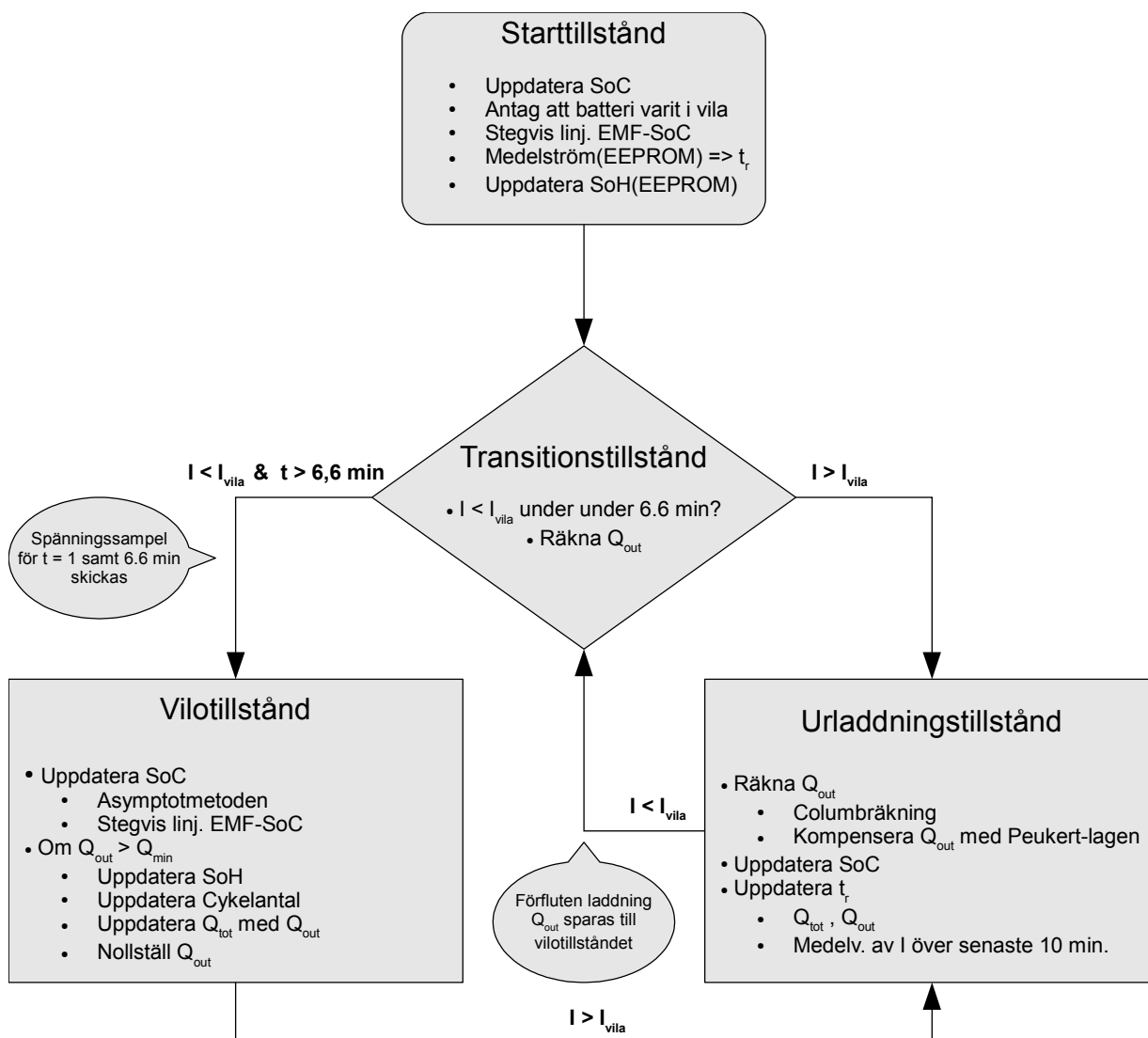
Figur 7 4: Kretskort med monterade komponenter.

1. Header för spänningsmatning 12 Volt.
2. PTC-motstånd för att begränsa strömtillförsel till systemet och likriktardiod för att motverka att systemet tar skada ifall spänningsregulatorn monteras felvänd.
3. Spänningsregulator MIC29150 Micrel.
4. Header för inkoppling av shuntspänningen.
5. Strömförstärkare LT6100 från Linear Technology.
6. Extern resonator CSTCC10M0G53-R0 från Murata.
7. Programmeringsheader.
8. CAN-tranceiver MCP2551 från Microchip. (Ej monterad på bild.)
9. Header för anslutning av CAN och RS-232 kommunikation.
10. EEPROM 24LC128 från Microchip.
11. Mikroprocessor dsPIC30F4013 från Microchip.
12. RS-232 tranceiver MAX232 från Maxim.
13. Debugheader.

8 ALGORITMUTVECKLING

Batteriövervakningssystemets algoritm har till uppgift att med hjälp av spännings- och strömmätning uppskatta parametrarna laddningsgrad (SoC), återstående körtid (t_r) samt hälsostatus (SoH) hos ett batteri. Algoritmen använder sig av olika metoder för att beräkna parametrarna beroende på hur batteriet används. För att algoritmen skall kunna avgöra vilken metod som skall användas är den uppbyggd av olika tillstånd. Mellan dessa tillstånd finns villkor för att transitering ska ske.

Utvecklingen av algoritmen har grundat sig i den kunskap som inhämtades under inläsning av litiumjonbatteriets kemiska och elektriska egenskaper. Med dessa kunskaper har en kombination av olika metoder använts för att uppskatta parametrarna. Mer om litiumjonbatteriets uppbyggnad kan läsas i avsnitt 2.3 : *Teknisk bakgrund: Litiumjonbatteriet*. Mer teorin till algoritmutvecklingen kan läsas i avsnitt 2.1 : *Teknisk bakgrund: Algoritmer och metoder*. Nedan presenteras ett flödesschema för algoritmen:



Figur 8 1: Flödesschema för algoritmen.

8.1 Starttillstånd

Starttillståndet är det första tillståndet som körs när systemet startar eller startas om. Detta kan ses som en initieringsprocess av algoritmen. I starttillståndet hämtas systeminformation från flashminnet. Till denna systeminformation hör gränsvärdet för standby-strömmen, minsta acceptabla urladdning för att uppdatera hälsostatus, maximala kapacitetsdifferensen som är acceptabel vid uppdatering av hälsostatus, vald batterikemi, antalet battericeller, nominell kapacitet, Peukert-koefficienten, verkliga kapaciteten (Q_{TOT}) samt medelströmmen under körning.

Laddningsgraden uppdateras genom polspänningen på batteriet. Metoden som används är stegvis linjarisering av förhållandet mellan EMF och laddningsgrad. För att denna uppskattning ska vara rimlig antas det vid start av systemet att batteriet varit i vila, eftersom det endast är då som polspänningen överensstämmer med EMF-spänningen.

Efter att parametrarna har uppdaterats sker en ovillkorlig transitering till transitionstillståndet.

8.2 Transitionstillstånd

Transitionstillståndet används för att avgöra om strömmen från batteriet har varit under gränsen för standby-ström i 6,6 minuter. Detta är den tid som asymptotmetoden kräver för att kunna uppskatta EMF-spänningen i vilotillståndet.

Om villkoret är uppfyllt lämnas transitionstillståndet och algoritmen fortsätter med vilotillståndet. I annat fall sker transitering till urladdningstillståndet.

8.3 Vilotillstånd

När algoritmen kommer till vilotillståndet är det känt att batteriet är i vila. Laddningsgraden uppdateras via stegvis linjarisering av förhållandet mellan EMF och laddningsgrad. EMF-spänningen uppskattas med hjälp av asymptotmetoden som använder sig av två sampels vid tiderna 0 samt 6.5 minuter. För mer detaljerad information om asymptotmetoden hänvisas till avsnitt 2.1.1 : *Teknisk bakgrund: Algoritmer och metoder: Direkta mätningar.*

Om den förflutna laddningen sedan tidigare vila, är större än ett gränsvärde, uppdateras hälsostatus. Uppdateringen är möjlig genom att använda skillnaden i laddningsgrad mellan de två vilotillstånden tillsammans med den förflutna laddningen och den nominella batterikapaciteten enligt sambandet:

$$SoH = \frac{100}{SoC_{start} - SoC_{slut}} \cdot \frac{Q_{OUT}}{Q_{NOM}} \quad \text{Formel 8.1.}$$

Om strömmen överskrider gränsen för standby-strömmen transiterar algoritmen till urladdningstillståndet, eftersom batteriet då inte längre befinner sig i vila.

8.4 Urladdningstillstånd

I urladdningstillståndet uppdateras laddningsgraden med hjälp av Coulomb-räkning. För mer information om Coulomb-räkning hänvisas till avsnitt 2.1.2 : *Teknisk bakgrund: Algoritmer och metoder: Bokförande system*. Detta eftersom batteriets EMF-spänning inte längre kan uppskattas. Genom att tillgänglig laddning, vid transitering till urladdningstillståndet, är känt kan SoC beräknas. Räknad laddning kompenseras för Peukert-effekten. För mer information om Peukert-effekten se avsnitt 2.3.3 : *Teknisk bakgrund: Litiumjonbatteriet: Icke-ideala faktorer*.

Med hjälp av hälsostatusen och det uppdaterade värdet för laddningsgraden tillsammans med medelströmmen beräknas återstående körtid (t_r).

När strömmen underskrider gränsen för standby-ström återgår algoritmen till transtitionstillståndet.

9 KOMMUNIKATION

9.1 CAN

Avsnittet behandlar hur sändning av data via CAN sker. För mer information om CAN-standarden hänvisas till avsnitt 2.8.1 : *Teknisk bakgrund: Kommunikation: CAN*.

CAN-kommunikationen i radiobilen använder sig av extended data frame, vilket innebär att varje meddelande som sänds på bussen föregås av en 29 bitars identifierare. Identifieraren består av klass-id (4 bit), aktivitetstyp (10 bitar), aktivitets-id (6 bitar) och sändar-id (9 bitar). Identifieraren har angivits utgående från en standard som presenteras i [Ref. ⁴³]. Standarden innebär att de 4 första bitarna i identifieraren beror av vad för slags data det är som skickas. Batteriövervakningssystemet sätts i enlighet med standarden till att vara i klassen för sensorer, vilket motsvarar 0x02. Nästföljande 9 bitar av identifieraren ger vilken del av systemet som sänder. Denna identifierare är satt med hänsyn till prioritet vad gäller sändning på bussen. Batteriövervakningssystemet är här satt till en låg prioritet. Därefter följer 6 bitar som identifierar olika typ av data som sänds från noden, enligt den rådande konfigurationen är dessa bitar inte en del av arbitreringen utan fungerar mer som ett sätt för övriga noder att urskilja olika typer av data på bussen. De sista 9 bitarna ger ytterligare möjlighet att identifiera fler olika meddelanden på bussen.

Identifierare

Klass-id				Aktivitetstyp										Aktivitets-id						Sändar-id									
0	0	1	0	0	1	1	1	1	1	1	1	1	1	x	x	x	x	x	x	0	0	0	0	0	0	0	0	1	0

C1TX0SID

SID<10:6>								SID<5:0>						TXIDE	
0	0	1	0	0	0	0	0	1	1	1	1	1	1	0	1

C1TX0EID

EID<17:14>								EID<13:6>							
1	1	1	x	0	0	0	0	x	x	x	x	x	0	0	0

C1TX0DLC

EID<5:0>								DLC<3:0>							
0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0

Figur 9.1: 29 bitars identifierare samt hur registrena sätts i samband med sändning av meddelande.

Identifieraren anges genom skrivning till tre register, C1TX0SID, C1TX0EID och C1TX0DLC. Där det sistnämnda registret även används till att ange hur många bytes som ska sändas åt gången. Vid anrop av CAN funktionen skrivs identifieraren till de ovan nämnda registren och meddelandet skrivs till transmitt buffern, C1TX0B1. Därefter startas sändningen genom att biten TXREQ i registret C1TX0CON sätts till true. Därefter väntar funktionen på att systemet ska sätta TXREQ till

false vilket inträffar vid lyckad sändning. Biten TXIDE i register C1TX0SID sätts till true för extended data frame.

Nedan presenteras den information som delges via CAN-kommunikation en gång var femte sekund samt informationens aktivitets-ID.

Tabell 9 .1: Parametrar som skickas via CAN-kommunikation samt dess aktivitets id.

Parameter	Beteckning	Enhet / Värden	Uppdateras	Aktivitets-ID
Spänning	U	Volt	Alltid (Visar senaste sekunden)	0x00
Ström	I	Ampere	Alltid (Visar senaste sekunden)	0x01
Laddningsgrad	SoC	%	Alltid	0x04
Hälsostatus	SoH	%	Mellan transiteringar till vilotillståndet under vissa villkor	0x05
Återstående körtid	t_R	Sekunder	När I_{AVG} är större än gränsströmmen för vila	0x07
Status	Status	(5) The battery is charged (> 75 % SoC) (4) The battery is partially charged (> 50 % SoC) (3) The battery is almost discharged (> 25 % SoC) (2) The battery is very discharged (< 25 % SoC) (1) The battery has reached it's end of life and should be replaced. (SoH < THR)	Alltid	0x06

9.2 RS-232

Avsnittet behandlar hur sändning av data via RS-232 sker. För mer information om RS-232-standarden hänvisas till avsnitt 2.8.2 : *Teknisk bakgrund: Kommunikation: RS-232*.

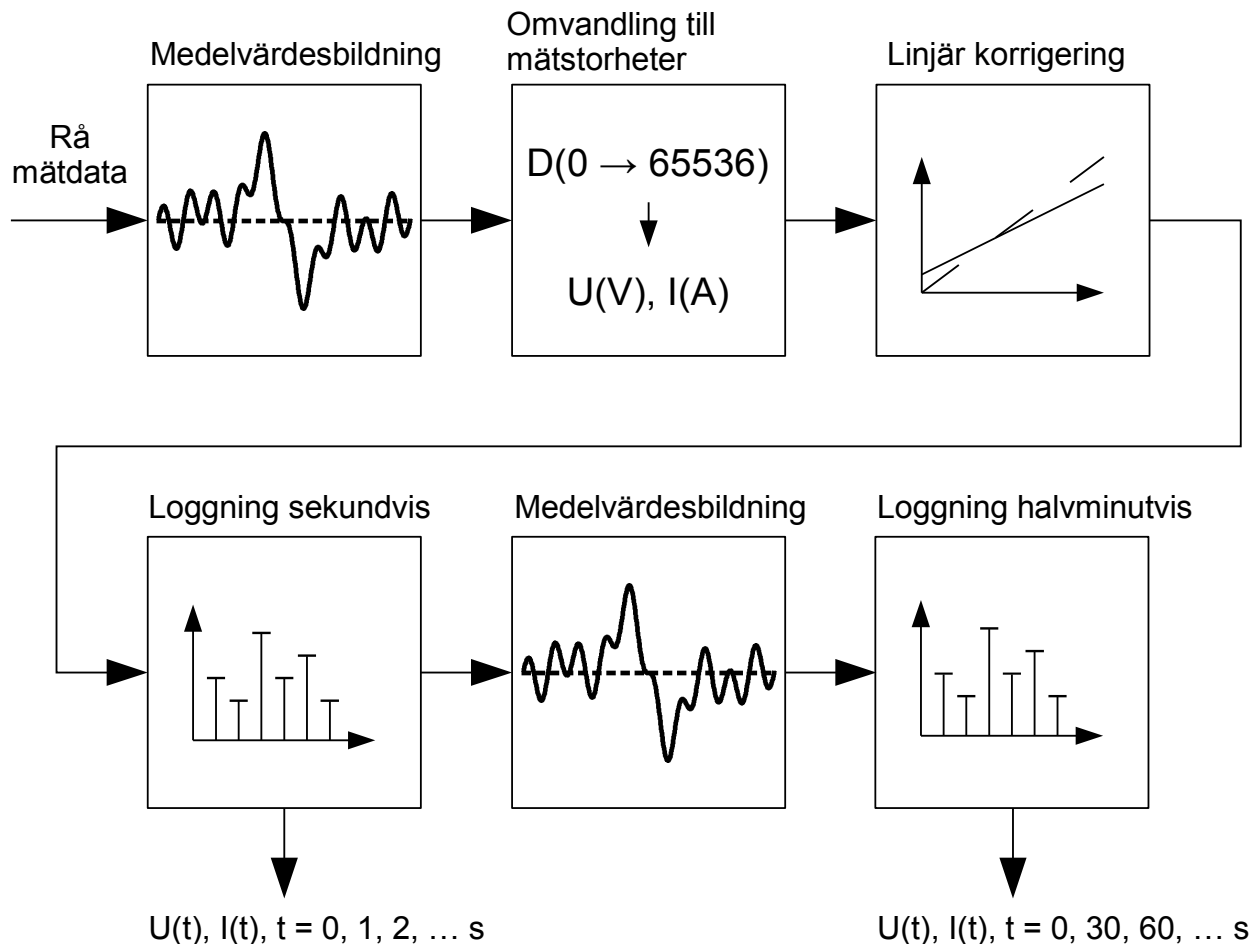
Kommunikation via RS-232 används av batteriövervakningssystemet för att ge detaljerad information om batteriet som övervakas. Det finns även en meny där parametrar som används av algoritmen går att konfigurera. Dessutom finns möjligheten att via denna meny ladda ner loggad data. Nedan presenteras den information som delges via RS-232-kommunikation var femte sekund.

Tabell 9 .2: Parametrar som skickas via RS-232-kommunikation.

Parameter	Beteckning	Enhet / Värdet	Uppdateras
Tillstånd	State	Starttillstånd (Start) Transitionstillstånd (Transition) Vilotillstånd (Equilibrium) Urladdningstillstånd (Discharge)	Alltid
Spänning	U	Volt	Alltid (Visar senaste sekunden)
Lyckade sampels av spänningen	S _U	Sampels/sekund	Alltid (Visar senaste sekunden)
Ström	I	Ampere	Alltid (Visar senaste sekunden)
Lyckade sampels av strömmen	S _I	Sampels/sekund	Alltid (Visar senaste sekunden)
Medelström under körning de senaste 10 minuterna	I _{AVG}	Ampere	När medelströmmen är över gränsströmmen för vila
Peukert-kompenserad ström	I _{PEUKERT}	Ampere	Alltid (Visar senaste sekunden)
Laddningsgrad	SoC	%	Alltid
Hälsostatus	SoH	%	Mellan transiteringar till vilotillståndet under vissa villkor
Nominell batterikapacitet	Q _{NOM}	Coulomb	Kan endast ändras i meny
Verklig batterikapacitet	Q _{TOT}	Coulomb	Mellan transiteringar till vilotillståndet under vissa villkor
Tillgänglig batterikapacitet	Q _{AVA}	Coulomb	Mellan transiteringar till vilotillståndet under vissa villkor
Genomfluten laddning	Q _{OUT}	Coulomb	I transitions- och urladdningstillståndet
Återstående körtid	t _R	Sekunder	När I _{AVG} är större än gränsströmmen för vila
Vilotid	t _I	Sekunder	Under transitions- och vilotillståndet
Status	Status	Se CAN-kommunikation	Alltid

10 SIGNALBEHANDLING

Batteriövervakningssystemet använder digital signalbehandling för att erhålla värden för spänning och ström. Signalbehandlingen består av ett antal olika steg som var och en gör olika operationer på datan. Figuren nedan visar den väg som data flyter genom systemet.



Figur 10.1: Flödesschema för signalbehandlingen i mätsystemet.

Det första steget involverar *medelvärdesbildning* av rå mätdata från analog till digital-omvandlingarna. Medelvärdesbildningen i detta steg görs för att uppfylla samplingsteoremet samt för att erhålla högre upplösning hos signalen. För mer information om teorin som används hänvisas till avsnittet 2.5 : *Teknisk bakgrund: Sampling, samplingsteoremet och antivikningsfilter*. Vidare hänvisas till avsnitt 5.1.4 : *Konfiguration av elektriska kretsar: Mikrokontroller, Microchip dsPIC30F4013: Konfiguration av analog till digital-omvandlaren* samt 5.1.4 : *Konfiguration av elektriska kretsar: Mikrokontroller, Microchip dsPIC30F4013: Konfiguration av räknare*, för mer information om konfigurationen om systemets sampling. Samplingen av spänning och ström sker med frekvensen 500 Hz. Medelvärdebildningen sker en gång per sekund och sker således nominellt över 500 värden. Resultatet av detta steg är ett nytt datavärde för spänning respektive ström en gång i sekunden. Detta värde ger emellertid inte direkt värden för spänning och ström i Volt och Ampere. Därför behövs nästföljande steg. Implementationen av medelvärdesbildningen är gjord så att avbrottsrutinen för analog till digital-omvandlingarna adderar den senaste 12-bitars sampelns $D_{i,12\text{-bit}}$

till en summavariabel D_{SUM} . En gång i sekunden hämtas sedan värdet i denna variabel och ett 16-bitars värde $\bar{D}_{\text{AVG},16\text{-bit}}$ beräknas enligt:

$$\bar{D}_{\text{AVG},16\text{-bit}} = 2^4 \cdot \frac{1}{500} \cdot D_{\text{SUM}} = 2^4 \cdot \frac{1}{500} \sum_{i=1}^{500} D_{i,12\text{-bit}} \quad \text{Formel 10 .1.}$$

Det andra steget i signalbehandlingen är *omvandling till mätstorheter*. I detta steg omvandlas rå mätdata till spänning i Volt och ström i Ampere. Omvandlingen sker till så kallade fraktionella tal så att spänning och ström kan användas vid beräkningar samt skrivas ut. För mer information om fraktionella tal se avsnitt 2.2 : *Teknisk bakgrund: Fixpunktsaritmetik*. Implementationen av omvandlingen till mätstorheter är gjord genom en funktion som dividerar mätdata med en koefficient α för spänning och β för ström enligt:

$$\alpha = \frac{2^{Q_I}}{U_{\text{MAX}}} = \frac{2^8}{30 \text{ Volt}} \approx 8,533 \quad , \quad \text{Formel 10 .2.}$$

där Q_I är antalet bitar i det fraktionella talets heltalsdel samt U_{MAX} den maximala spänningen som kan mätas.

$$\beta = \frac{2^{Q_I}}{I_{\text{MAX}}} = \frac{2^8}{100 \text{ Ampere}} = 2,56 \quad , \quad \text{Formel 10 .3.}$$

där Q_I är antalet bitar i det fraktionella talets heltalsdel samt I_{MAX} den maximala strömmen som kan mätas.

Med de erhållna värdena för spänning och ström gäller det att dessa kommer att innehålla mätfel som kan bero på många orsaker i mätsystemet. I detta system används *linjär korrigering* för att justera mätdata efter resultatet av en kalibrering. Vanliga orsaker till mätfel är bland annat offset-spänningar och variationer inom toleranserna för resistorer och spänningsreferens. Ett enkelt sätt att förbättra precisionen i mätvärdena är således att justera för dessa fel med en offset m , samt en skalningskoefficient k , enligt det linjära sambandet:

$$y = kx + m \quad , \quad \text{där } x \text{ är indata (spänning eller ström) och } y \text{ är utdata.} \quad \text{Formel 10 .4.}$$

Detta ger naturligtvis inte en perfekt korrigering för all indata, men kan ändå avsevärt förbättra mätvärdena. För ett idealt mätsystem utan mätfel gäller att $k = 1$ samt $m = 0$. Se avsnitt 12.1 : *Testning: Kalibrering och test av spänningsmätning* samt 12.2 : *Testning: Kalibrering och test av strömmätning* för information om de resulterande k - och m -värdena för detta mätsystem samt resultatet av genomförd linjär korrigering. Implementationen av den linjära korrigeringen är gjord genom att använda det linjära sambandet ovan.

De värden för spänning och ström som erhålls efter den linjära korrigeringen sparas genom *loggning sekundvis*. Genom att göra detta blir det möjligt att se hur spänning och ström varierat sekundvis tillbaka i tiden. Denna information används av batteriövervakningssystemets algoritm. Implementationen sparar 30 sekunder tillbaka i tiden.

Ytterligare *medelvärdesbildning* görs av de senaste 30 sekundernas sparade data för spänning och ström. På detta sätt erhålls ett medelvärde för spänning och ström de senaste 30 sekunderna. Detta ligger till grund för nästa steg i signalbehandlingen.

Det sista steget i signalbehandlingen är *loggning halvminutvis*. Genom utdatan från det föregående steget kan medelvärdet av spänning och ström de senaste 30 sekunderna sparas en gång i halvminuten. Denna information används av batteriövervakningssystemets algoritm. Implementationen använder 30 värden och kan således återge data för spänning och ström de senaste 15 minuterna med upplösningen 30 sekunder.

11 LOGGNING TILL FLASHMINNE

I flashminnet lagras realtids information från körning. Denna information ska via val i ett menysystem gå att tanka ur då batteriövervakningssystemet kopplas till PC via RS-232. Tanken är att information som är viktig för systemets funktion även skall sparas undan på ett bestämt ställe på flashminnet. Detta för att systemet ska komma ihåg batteriets hälsostatus, medelström under de senaste 10 minutrarna samt parametrar för den fördefinierade batterikemin.

Storleken på flashminnet är 16 kB, antal skrivcykler detta motsvarar beror på storleken hos den data som ska lagras. I skrivande stund består körinformationen av 20 byte och den för systemet viktiga informationen av 26 byte. Vid körning kommer batteriövervakningssystemet att logga kördata var trettionde sekund, detta innebär med nuvarande konfiguration att flashminnet kan logga data i nästan 7 timmar innan minnet tar slut. I samma rutin som körs där kör-data loggas kommer även data med systeminformation att lagras undan, detta för att systemet ska fortsätta att logga systeminformation på rätt adress då systemets sätts igång igen efter att ha varit strömlöst. I den funktion som uppdaterar adressen i flashminnet finns en rutin som undersöker om en hel skrivning av kördata får plats, får den plats uppdateras adressen med storleken av kör-data, i annat fall sätts adressen till adress 0x0000 adderat med storleken på systeminformationen. I konfigurationsmenyn finns inställningar för hur långt tillbaka kördata ska loggas, oftast räcker det med att logga information från den senaste körcykeln.

11.1 Funktioner för läsning och skrivning av en byte

För att realisera den ovan beskrivna funktionen krävs ett antal funktioner. De mest grundläggande är *eeeprom_read_byte* och *eeeprom_write_byte* som läser respektive skriver en byte till flashminnet. I dessa funktioner genomförs det för hårdvaran specifika tillvägagångssättet för att skriva respektive läsa en byte från flashminnet. Funktionerna för att läsa och skriva en post till flashminnet kommer sedan att bygga på dessa funktioner.

11.2 Funktioner för läsning och skrivning av en post

Dessa funktioner har till uppgift att läsa respektive skriva en post till minnet genom att dela upp posterna i delar om en byte. Därefter läses respektive skrivs varje byte till minnet en efter en med hjälp av funktionerna *eeeprom_read_byte* och *eeeprom_write_byte*. Funktionerna *eeeprom_read_status_struct* och *eeeprom_write_status_struct* har samma funktion med den skillnaden att de läser och skriver posten för systeminformationen.

11.3 Funktioner för reset och formatering

Dessutom finns hjälpfunktioner för att återställa EEPROM-adressen till utgångsläget samt att formatera flashminnet. Vid formatering av flashminnet skrivs för närvarande strängslutstecken '\0' till dess samtliga positioner. Detta sker med funktionerna *reset_eeprom* samt *eeprom_format*.

11.4 Funktioner för lagring och utskrift av kördata

De tidigare beskrivna funktionerna används sedan till att lagra nuvarande kör-information i en post och skriva denna till flashminnet. Funktionerna används även till att skriva ut innehållet i flashminnet till konsolen via RS-232. Detta sker med funktionerna *eeprom_store_system_info* och *eeprom_send_system_info_rs232*.

11.5 Data som lagras på flashminne

Nedan presenteras den information som skrivs till flashminnet en gång var 30:onde sekund.

Tabell 11.1: Parametrar som skrivs till flashminnet.

Parameter	Beteckning	Enhet / Värdet	Uppdateras
Spänning	U	Volt	Alltid (Visar senaste sekunden)
Ström	I	Ampere	Alltid (Visar senaste sekunden)
Medelström under körning de senaste 10 minuterna	I_{AVG}	Ampere	När medelströmmen är över gränsströmmen för vila
Laddningsgrad	SoC	%	Alltid
Hälsostatus	SoH	%	Mellan transiteringar till vilotillståndet under vissa villkor
Nominell batterikapacitet	Q_{NOM}	Coulomb	Kan endast ändras i meny
Verklig batterikapacitet	Q_{TOT}	Coulomb	Mellan transiteringar till vilotillståndet under vissa villkor
Tillgänglig batterikapacitet	Q_{AVA}	Coulomb	Mellan transiteringar till vilotillståndet under vissa villkor
Genomfluten laddning	Q_{OUT}	Coulomb	I transitions- och urladdningstillståndet
Återstående körtid	t_R	Sekunder	När I_{AVG} är större än gränsströmmen för vila
Status	Status	(5) The battery is charged (> 75 % SoC) (4) The battery is partially charged (> 50 % SoC) (3) The battery is almost discharged (> 25 % SoC) (2) The battery is very discharged (< 25 % SoC) (1) The battery has reached it's end of life and should be replaced. (SoH < THR)	Alltid

12 TESTNING

12.1 Kalibrering och test av spänningsmätning

Spänningsmätningen behöver kalibreras för att de initiala fel som finns hos mätsystemet. Detta är fel som kan härledas till bland annat toleransen för resistansvärden i spänningsdelningen samt offset-spänningen in till analog-till-digitalomvandlaren.

Kalibreringsförfarandet inleds med att en känd spänning, mycket nära 0 Volt används som analog insignal till systemet. Den av mätsystemet uppmätta spänningen, mätfelet vid 0 Volt, är då den offset-spänning m_U som kan kalibreras för. Med denna känd och kalibrerad för skall sedan en spänning så nära 30 Volt som möjligt användas som analog insignal till systemet. Genom att jämföra det av mätsystemet uppmätta spänningen med den faktiska spänningen kan ett värde för skalningsfelet k_U beräknas. Dessa två värden, offset-spänningen och skalningsfelet, kan sedan med hjälp av makron programmeras in i systemet så att en bättre noggrannhet kan erhållas hos spänningsmätningen.

Resultatet av spänningskalibreringen ger följande värde för offset-spänningen m_U samt skalningsfelet k_U :

$$m_U = U_{\text{mät}} - U_{\text{visad}} = 0,10 \text{ Volt} - 0,07 \text{ Volt} = 0,03 \text{ Volt} \quad \text{Formel 12.1.}$$

, där $U_{\text{mät}}$ är den egentliga spänningen som mättes och U_{visad} den spänning som systemet visar.

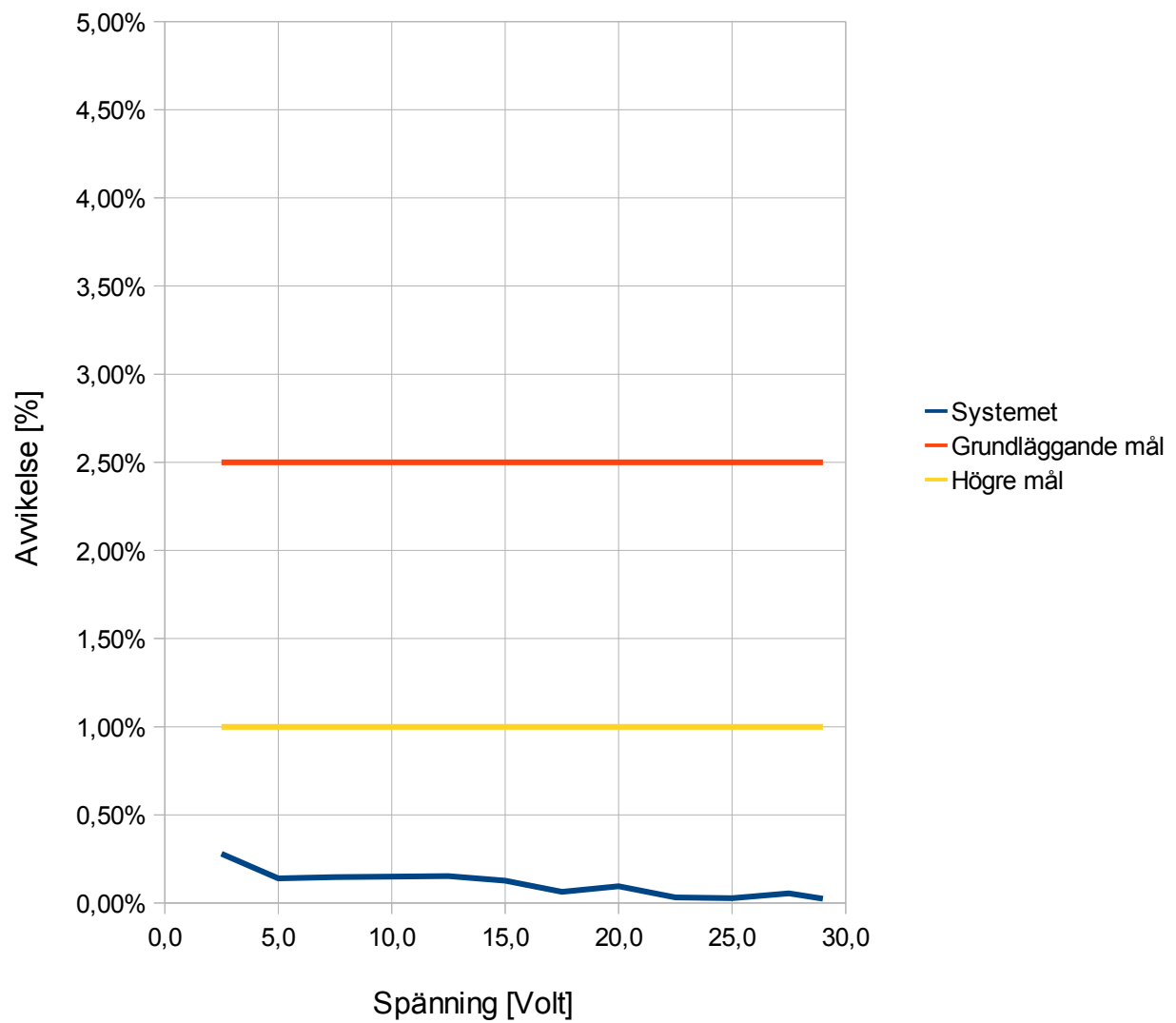
$$k_U = \frac{U_{\text{mät}}}{U_{\text{visad}}} = \frac{28,02 \text{ Volt}}{27,96 \text{ Volt}} \approx 1,002146 \text{ V/V} \quad \text{Formel 12.2.}$$

, där $U_{\text{mät}}$ är den egentliga spänningen som mättes och U_{visad} den spänning som systemet visar.

Slutligen skall ett test genomföras för att kontrollera noggrannheten hos den kalibrerade spänningsmätningen. För att genomföra dessa test finns en multimeter av modell Agilent U1242B med en uppmätt spänningstolerans om 0,01 % samt en ettårig specifikation om 0,11 % för 100 Volt-området. Testet kommer att genomföras för spänningar inom området 2,5 → 29 Volt. Vid dessa spänningar kommer resultatet att plottas som mätfelet för spänningsmätningen hos systemet i procent som funktion av spänning.

En genomförd test visar att spänningsmätningen fungerar mycket väl. Nedan presenteras ett diagram över resultatet av mät-testet. I diagrammet finns också markerat för gränserna att klara de grundläggande målen samt de högre målen med avseende på spänningsmätningen. Det går att urskilja att noggrannheten är sämre vid lägre spänningar och att det maximala felet är under 0,3 % vid en spänning om 2,5 Volt.

Test av spänningsmätning



Figur 12.1: Resultatet efter kalibrering av spänningsmätningen.

12.2 Kalibrering och test av strömmätning

Kalibreringen av strömmätningen behöver likt spänningsmätningen göras för att kompensera för initiala fel som finns hos mätsystemet. Detta fel kan härledas till bland annat offset-spänningen hos strömförstärkaren samt eventuellt skalningsfel som kan finnas hos denna. Vidare kan mikrokontrollerns analog-till-digitalomvandlare ha en offset-spänning såväl som ett skalningsfel.

Kalibreringsförfarandet för strömmätningen kommer att genomföras med en så kallad simulerad shunt-spänning. Detta eftersom det i labbet inte finns någon möjlighet att driva en konstant ström inom intervallet $0 \rightarrow 100$ Ampere. Detta är inget problem i sig eftersom spänningen från ströms hunten kan simuleras med hjälp av en enkel potentiometer. Detta kommer dock att resultera i att fel resistansvärdet för ströms hunten kommer att försummas, vilket kommer att försämra noggrannheten för det slutgiltiga systemets strömmätning. Toleransen för ströms hunten är 0,25 % vilket då blir den undre gränsen för vad systemet kommer att kunna mäta ström inom. Genom att först mäta en ström mycket nära 0 A kan offset-strömmen m_I beräknas. Efter att denna har beräknats och justerats för skall en ström mycket nära 100 A mätas. Det uppmätta värdet kommer då att ge skalningsfelet k_I . Dessa två värden, offset-strömmen och skalningsfelet, kan sedan med hjälp av makron programmeras in i systemet så att en bättre noggrannhet kan erhållas hos strömmätningen. Värt att notera är att båda dessa mätningar måste genomföras vid en common mode-spänning om 22,2 Volt.

Resultatet av strömkalibreringen ger följande värde för offset-strömmen m_I samt skalningsfelet k_U :

$$m_I = I_{\text{mät}} - I_{\text{visad}} = 1,0 \text{ A} - 0,875 \text{ A} = 0,125 \text{ A} \quad \text{Formel 12 .3.}$$

, där $U_{\text{mät}}$ är den egentliga spänningen som mättes och U_{visad} den spänning som systemet visar.

$$k_I = \frac{I_{\text{mät}}}{I_{\text{visad}}} = \frac{100,0 \text{ A}}{98,609 \text{ A}} \approx 1,014106 \text{ A/A} \quad \text{Formel 12 .4.}$$

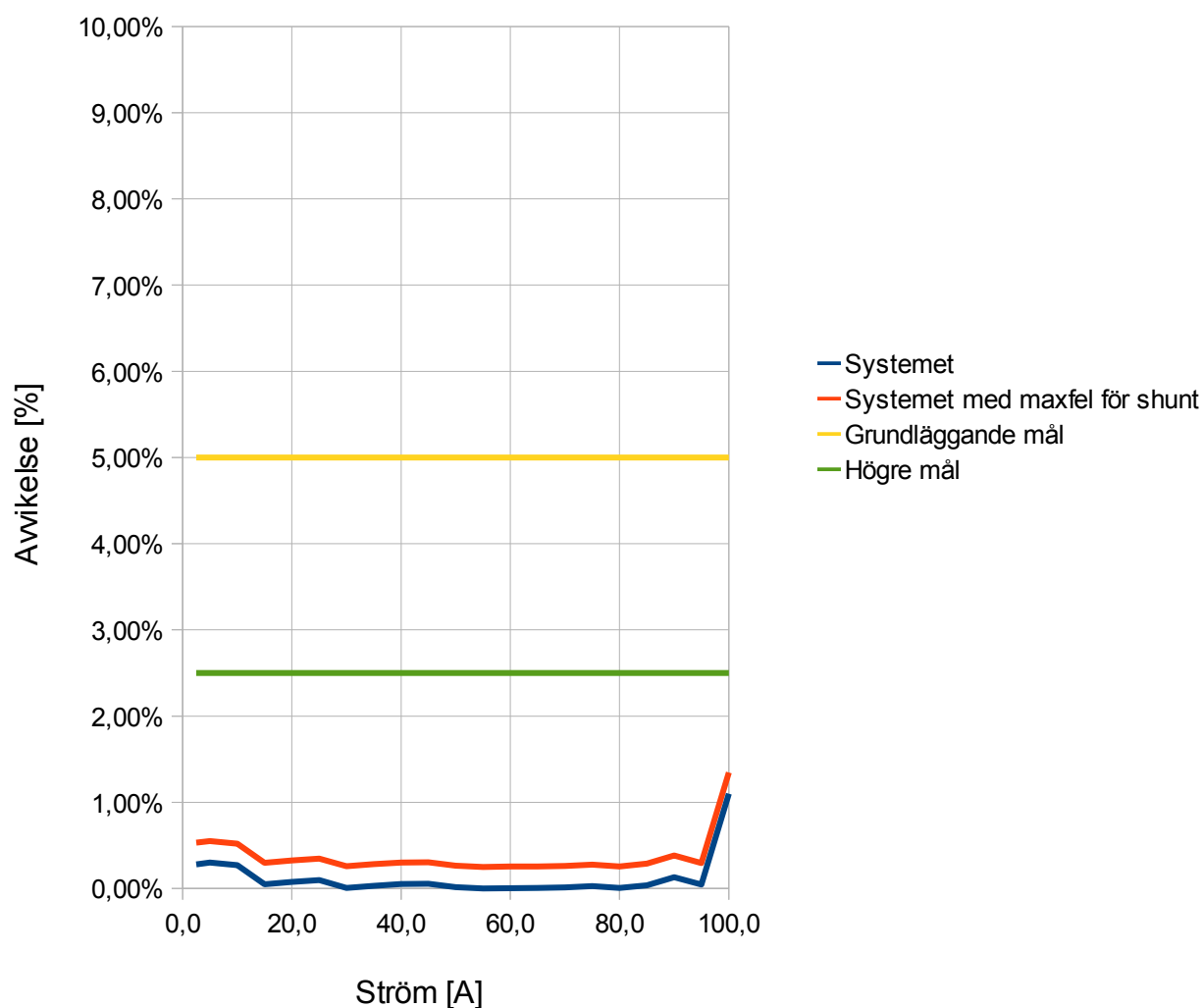
, där $I_{\text{mät}}$ är den egentliga strömmen som mättes genom en simulerad shuntspänning och I_{visad} den spänning som systemet visar.

Slutligen skall ett test genomföras för att kontrollera noggrannheten hos den kalibrerade strömmätningen. För att genomföra dessa test kommer samma multimeter som för kalibreringen av spänningsmätningen ovan att användas. Multimetern har en uppmätt spänningstolerans om 0,00 % samt en ettårig specifikation om 0,14 % för 1000 mV-området. Testet kommer att genomföras vid en simulerad shuntspänning om 2,5, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95 samt 100 mV vilket kommer att motsvara området $0 \rightarrow 100$ Ampere. Vid dessa spänningar kommer resultatet att plottas som mätfelet för strömmätningen hos systemet som en funktion av ström. En ytterligare linje kommer att presenteras med det maximala verkliga mätfelet om hänsyn till att endast simulerad shuntspänning används. Detta görs genom att toleransen för ströms hunten om 0,25 % adderas till mätfelet som funktion av ström. Ytterligare två referenslinjer som presenterar målet för mätfelet hos systemet kommer att finnas i diagrammet.

Nedan visas ett diagram över resultatet för strömmätningen efter genomförd kalibrering. Resultatet av strömmätningen visar att den maximala avvikelserna även med toleransen för strömshunten i beräkning är väl inom målen för systemet.

Test av strömmätning

Med simulerad shuntspänning



Figur 12.2: Resultatet efter kalibrering av strömmätningen.

12.3 Uppmätning av driftström

Driftströmmen hos systemet är den ström som erfordras för att köra systemet under normal drift. Denna är intressant för att veta vad modulen väntas förbruka i effekt i verkligheten. Testet genomförs genom att mäta strömmen till spänningsmatningen av modulen vid en inspänning om 12 Volt nominellt. Resultatet kommer att presenteras som en ström vars värde motsvarar modulens ungefärliga strömförbrukning vid normal funktion. Uppmätt ström vid 12 volts driftspänning är:

$$I_{DRIFT} \approx 28 \text{ mA} \Rightarrow P = UI = 12 \cdot 28 \text{ mA} \approx 0,34 \text{ W} \quad \text{Formel 12.5.}$$

12.4 Test av algoritmen

För att testa algoritmen som är implementerad i systemet genomförs olika test för att verifiera och presentera prestanda för olika funktioner hos algoritmen. Testerna kommer att ligga till grund för en övergripande bedömning av systemets prestanda samt eventuellt påvisa fel och/eller förslag på förbättringar. Med hjälp av de erhållna resultaten kan också specifikationer för systemet bestämmas och jämföras mot kravspecifikationen för projektet.

Tester som genomförts är partiellt urladdningstest vid konstant ström samt ett fullt urladdningstest vid konstant ström.

Det partiella urladdningstestet genomförs på ett fulladdat batteri. Batteriets hälsostatus uppskattas till 91,7% genom tidigare körcykel. Testet påbörjas genom att en tredjedel av batteriets kapacitet laddas ur. Därefter lämnas batteriet att vila i 6,5 minuter så att transitering till vilotillståndet möjliggörs. Förfarandet upprepas sedan tills batteriet är tomt. Vid tomt batteri inväntas återigen transitering till vilotillståndet, därefter avslutas testcykeln.

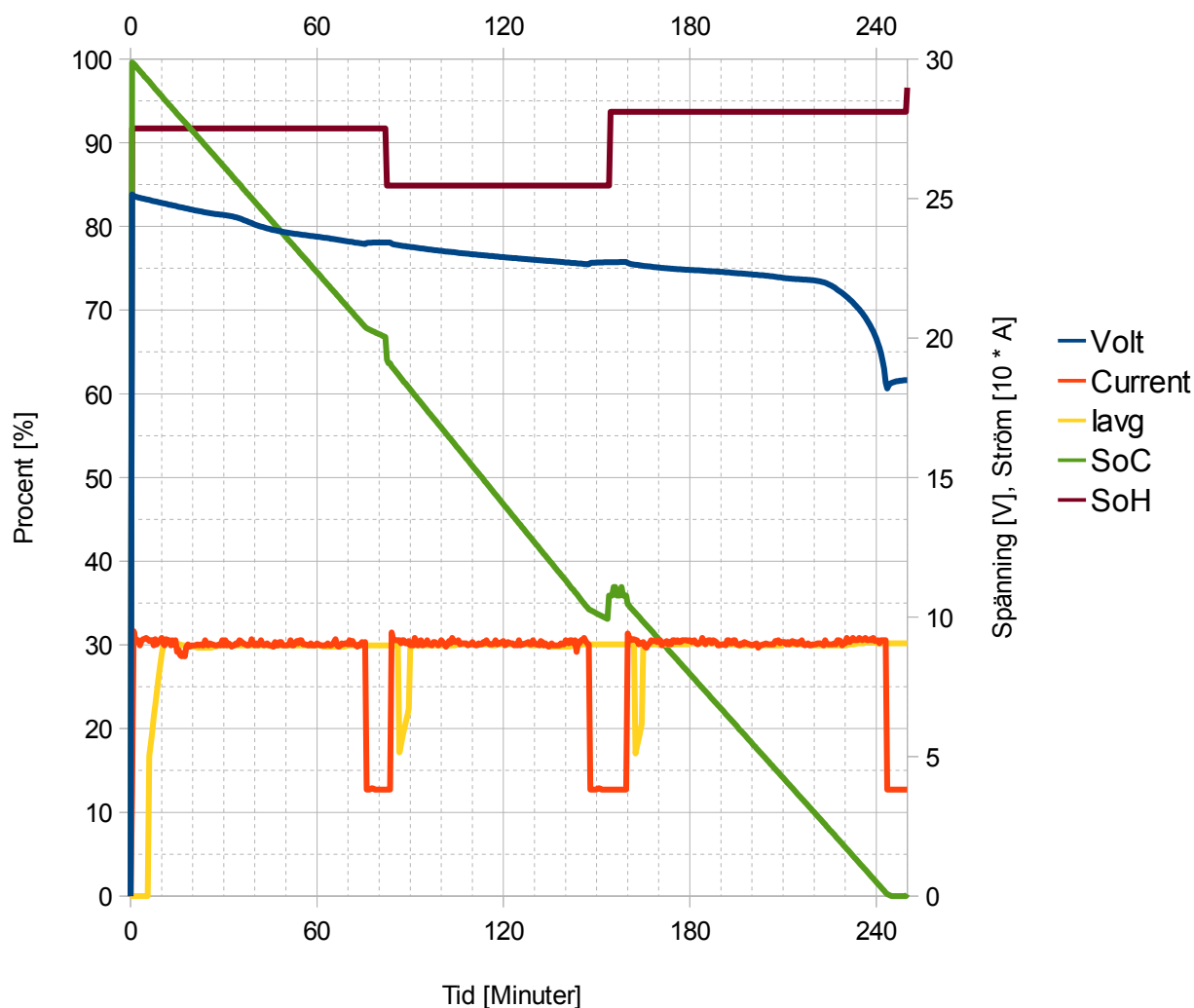
Det fulla urladdningstestet utförs även det med ett fulladdat batteri men med en uppskattad hälsostatus om 90 %. Denna hälsostatus kan ställas in via batteriövervakningssystemets menysystem. För mer information om hur systemets parametrar kan konfigureras se bilaga *Användningsinformation*. Skillnaden mellan detta testet och det förra är att algoritmen inte tillåts att transitera till vilotillståndet under test-cykeln. Vid tomt batteri sker transitering till vilotillståndet där laddningsgrad och hälsostatus uppdateras. Denna uppskattning av hälsostatus kan anses vara mycket nära den verkliga då det i detta skedet av testcykeln finns värde på hur stor laddning som flutit ur batteriet under en fullständig urladdning.

Syftet med de två testerna är att avgöra hur noggrant systemet kan uppskatta laddningsgrad då transitering till vilotillståndet ej tillåts. Detta resultat kan sedan jämföras med resultatet från testcykeln med regelbunden transitering till vilotillståndet. Ur resultaten ska det även gå att analysera hur väl uppskattningen av hälsostatus stämmer överens med verkligheten.

Värt att notera är att både det partiella och det fulla urladdningstestet genomfördes med en konstant urladdningsström om 1 A. Anledningen till detta är att detta var den maximala strömmen laddnings/urladdnings utrustningen tillåter.

Resultat av partiell urladdning

1 Ampere konstant ström



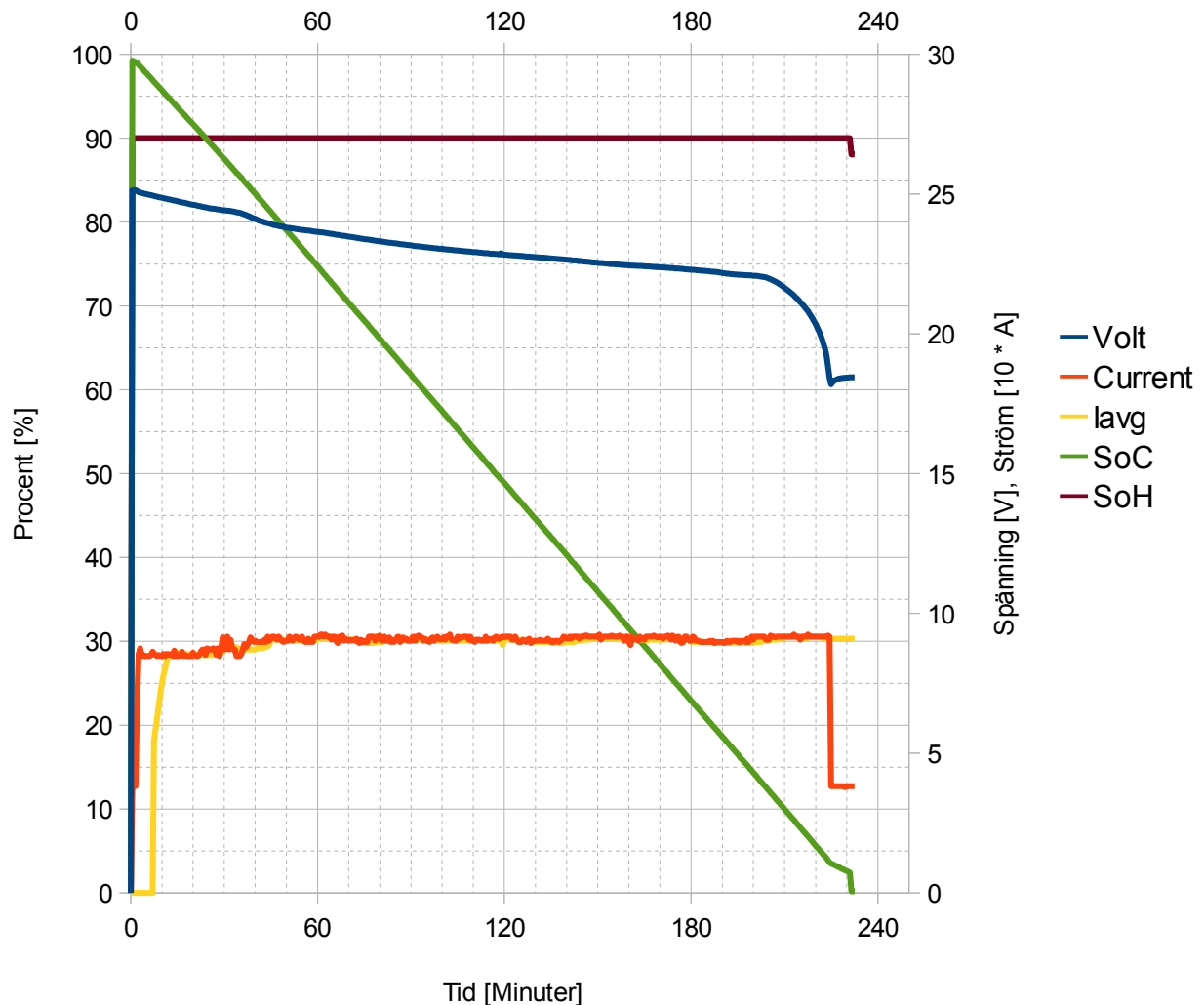
Figur 12.3: Resultatet av det partiella urladdningstestet.

Ur diagrammet över det partiella urladdningstestet ovan kan transitering till vilotillståndet tydligt urskiljas. Vid dessa transiteringar uppdateras laddningsgrad och hälsostatus. I diagrammet går det att urskilja hur laddningsgrad och hälsostatus justerades ner vid det första vilotillståndet och justerades upp vid det andra. Dessa värden hänger naturligt ihop eftersom laddningsgrad är ett procentuellt värde av hur stor del av den nominella kapaciteten som finns tillgängligt i batteriet.

Testcykeln visar på en oregelbundenhet i uppdateringen av hälsostatusen. Värt är även att notera att felet i återstående körtid vid testcykeln var endast 7 sekunder. Det goda resultatet beror på att batteriet laddas ur med en konstant ström, vilket gör att medelströmmen är ett mycket exakt värde på den verkliga strömmen som dras ur batteriet.

Resultat av full urladdning

1 Ampere konstant ström



Figur 12.4: Resultatet av det fulla urladdningstestet.

I diagrammet över det fulla urladdningstestet är två punkter av extra stort intresse. Dessa är felet i laddningsgrad vilket kan utläsas som kvarvarande laddningsgrad vid tomt batteri, och det uppdaterade värdet av hälsostatusen vilket går att utläsa i slutet av testcykeln.

Av testerna går det att dra en del slutsatser om hur väl batteriövervakningssystemet fungerar. Testerna ger dock endast en ungefärlig uppfattning om systemets prestanda då strömmen vid mer realistisk belastning är större och mer oregelbunden. Tillsammans visar testen att både Coulomb-räkning och asymptot-metoden ger bra värden på laddningsgraden. Anledningen till felet i uppskattning av laddningsgrad vid det fulla urladdningstestet är att uppskattad hälsostatus inte överensstämmer med batteriets verkliga hälsostatus. Skulle testet återupprepas med samma batteri och den uppdaterade hälsostatusen skulle antagligen resultatet ha blivit bättre. En slutsats som går att dra av testerna är att en större del av batteriet bör laddas ur innan en uppdatering av hälsostatusen genomförs. I övrigt fungerar batteriövervakningssystemet till belåtenhet.

13 SPECIFIKATIONER

Nedan presenteras en tabell över specifikationerna för systemet. Dessa är framtagna utifrån komponenternas specifikationer, designval under utvecklingen samt resultat efter test. Om inget annat anges gäller dessa parametrar vid en matningsspänning om 12 Volt samt en CM-spänning till spännings- och strömmätningen om 22 Volt.

Parameter	Min	Typ	Max	Enhet
Matningsspänning	6,0	12	35	Volt
Strömförbrukning	–	28	–	mA
Effektförbrukning	–	0,34	–	W
Spänningsmätning	~ 0	22	30	Volt
Tol. Spänningsmätning	–	0,25	~ 0,50	%
Strömmätning	~ 0	–	100	A
Tol. Strömmätning	–	0,50	~ 1,5	%
Tol. Laddningsgrad	–	~ 5	–	%
Tol. Hälsostatus	–	~ 10	–	%
Tol. Återstående körtid	–	~ 5	–	%
Samplingsfrekvens	–	~ 500	–	Hz
Loggningstid	0	–	360	Minuter
Tid mellan loggningar	–	30	–	Sekunder/Post
Instruktionsfrekvens	–	2,5	–	MHz
Nominell kapacitet	0	–	65 535	Coulomb

Parametrar specificerade med tilde [~] är uppskattade eller cirka-värden.

14 SLUTSATS OCH DISKUSSION

Projektet resulterade i en modul för batteriövervakning som kommunicerar via CAN. Modulen övervakar och uppdaterar kontinuerligt batteriets laddningsgrad och hälsotillstånd. Under körning loggas kördata till flashminne för senare urtankning och analys. Vid uppstart av systemet finns möjlighet att göra systeminställningar via en meny. Här kan det bland annat göras inställningar för aktuell batterikemi, antal celler, uppskattat värde på batteriets hälsotillstånd, övergångsvilkor för att uppdatera hälsotillstånd och inställningar för flashminnet såsom loggningstid och reset. Vid körning skickas även kördata ut via RS-232 vilket ger en möjlighet att följa systemets prestanda i realtid när bilen är upphängd i testbänk.

Vid början av projektet formulerades en kravspecifikation för systemet. Denna har legat till grund för val av komponenter. Ett av de högre målen i projektet är att kunna uppskatta batterikapaciteten med en noggrannhet om 5%. Med detta mål i åtanke anpassades ström- och spänningsmätningen till att vara så noggrann som möjligt. Detta resulterade i att målen för ström- och spänningsmätningen uppfylls med god marginal. Tester visar på ett system med mycket god precision.

Enligt resonemanget ovan uppfyller systemet mycket väl de mål som utformades vid projektets inledning. Dock finns där några saker som skulle kunna ha lösts på ett bättre sätt.

En av svagheterna med systemet har med uppskattningen av batteriets hälsotillstånd att göra. I diagrammet från det partiella urladdningstestet varierar uppskattningen av hälsostatusen kraftigt mellan vilotillstånden. Ett fel som bör avhjälpas genom att kravet på genomfluten laddning (Q_{out}), innan en uppdatering sker, ökas. Förslag till förbättring vid kommande mjukvarurevisioner är att vid uppdatering av hälsostatus, medelvärdesbilda mellan nytt och gammalt värde. På detta sätt minskar påverkan av en felaktigt beräknad hälsostatus.

I systemet delar ström- och spänningsmätningen ledning in till kretskortet. Eftersom spänningen via denna ledning delas ner via en spänningsdelning till jord, kommer en ström att gå denna väg. Denna ström ger ett spänningsfall om 2,2 mV över den PTC-resistorn som sitter monterad på strömshunten. Detta spänningsfall resulterar i att en offset-ström om 2,2 A detekteras av batteriövervakningssystemet. I det aktuella systemet löstes detta problem genom att även PTC-resistorn på shuntens nerspänningssida kopplas till jord via en 60 k Ω resistor. Spänningsfallet från uppspännings och nerspänningssidan tar då ut varandra. En bättre lösning på detta problem hade varit att konstruera systemet med separata ledningar för ström- spänningsmätning.

I rapporten lämnar test- och verifieringsdelen övrigt att önska. Detta eftersom testuppställningen endast möjliggjorde urladdningstester med 1 A. För att kunna testa batteriövervakningssystemet på allvar skulle en konstlast ha behövts. De specifikationer som rör laddningsgrad, hälsotillstånd och återstående körtid är därför grovt uppskattade värden.

Frågan som projektet ger svar på är huruvida det är möjligt, att med 10 veckors arbete, utveckla en modul som i stora drag kan uppskatta laddningsgrad, återstående körtid och hälsostatus för ett batteri i en elbil. Vår bedömning utifrån de tester som genomförts är att det går. Exakt hur bra vårt system presterar under mer verkliga förhållanden återstår dock att se. För mer utförliga tester krävs tillgång på konstlast.

REFERENSER

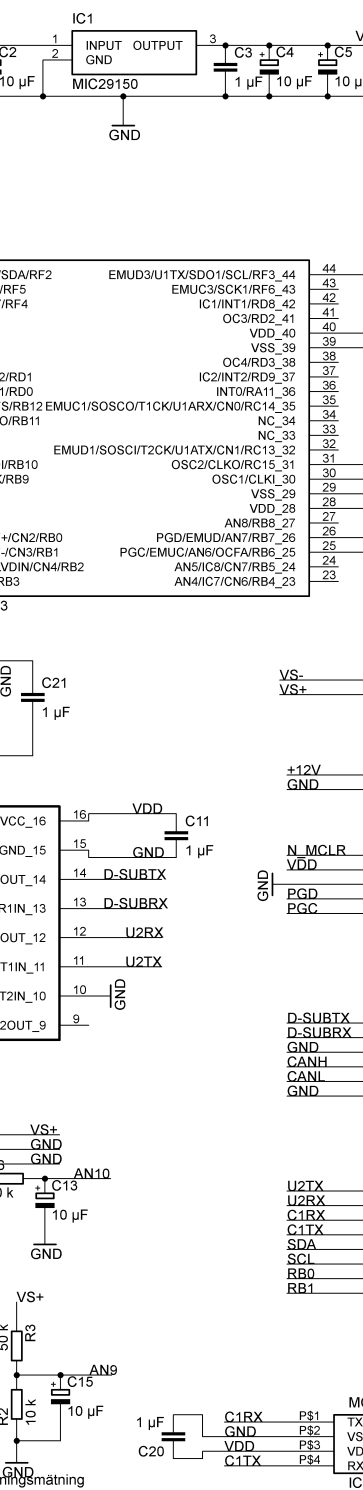
- 1 V. Pop, H.J. Bergveld, D. Danilov, P.P.L. Regtien and P.H.L. Notten: *Battery Management systems, Accurate State-of-Charge Indication for Battery-Powered Applications*. 2008.
- 2 Bengt Molin: *Analog elektronik*: Pozkal, Poland 2009
- 3 Erick L. Obestar: *Fixed-Point Representation & Fractional Math*, Oberstar Consulting, 2007.
- 4 Gold Peak Industries (Taiwan): *Lithium Ion Technical Handbook*, 2000.
- 5 How to Prolong Lithium-based Batteries, Battery University, 2010,
http://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries (Acc 2012-03-25).
- 6 *New Nanowire Battery Holds 10 Times The Charge Of Existing Ones*, ScienceDaily, 2007,
<http://www.sciencedaily.com/releases/2007/12/071219103105.htm> (Acc 2012-03-27).
- 7 Michael Root: *The TAB Battery Book*, McGraw-Hill, sid 175-182, 2011.
- 8 *Lithium-based Batteries*, Battery University, 2010, http://batteryuniversity.com/learn/article/lithium_based_batteries (Acc 2012-03-21).
- 9 *Is Lithium-ion the Ideal Battery?*, Battery Univerity, 2010,
http://batteryuniversity.com/learn/article/is_lithium_ion_the_ideal_battery (Acc 2012-03-25).
- 10 *Li-polymer Battery: Substance or Hype?*, Battery University, 2010,
http://batteryuniversity.com/learn/article/the_li_polymer_battery_substance_or_hype (Acc 2012-03-25).
- 11 Thomas L. Martin, Daniel P. Siewiorek: *Non-ideal Battery Properties and Low Power Operation in Wearable Computing, Institute for Complex Engineered Systems*, Carnegie Mellon University, 1999.
- 12 *Calculating the Battery Runtime (Peukert Law)*, Battery University, 2010,
http://batteryuniversity.com/learn/article/calculating_the_battery_runtime (Acc 2012-03-25).
- 13 *Charging litium-ion*, Battery University, 2010,
http://batteryuniversity.com/learn/article/charging_litium_ion_batteries (Acc. 2012-03-26)
- 14 *Protection circuits*, Battery University, 2010,
http://batteryuniversity.com/learn/article/safety_circuits_for_modern_batteries (Acc. 2012-03-26)
- 15 *Safety concerns with Li-ion*, Battery University, 2010,
http://batteryuniversity.com/learn/article/safety_concerns_with_li_ion (Acc. 2012-03-26)
- 16 Lars Bengtsson: *Elektriska mätsystem och mätmetoder*, 2003.
- 17 Warren Pettgrew: *Current Sensor Selection for Demanding Applications*, Ranztec Sensors, 2008.
- 18 Peter Abiodun Bode: *Current measurement applications handbook*, Zetex, 2008.
- 19 Wikimedia Commons, <http://commons.wikimedia.org/wiki/File:Shuntresistor50A.jpg> (Acc 2012-04-02).

- 20 Wikimedia Commons, <http://commons.wikimedia.org/wiki/File:HallEffCurrentSense.jpg> (Acc 2012-04-02).
- 21 Lars Bengtsson: *Elektriska mätsystem och mätmetoder*, sid 5, 2003.
- 22 Lars Bengtsson: *Elektriska mätsystem och mätmetoder*, sid 178, 204-205, 2003.
- 23 Bonnie C. Baker: *Anti-Aliasing, Analog Filters for Data Acquisition Systems*, Microchip Technology Inc, 1999.
- 24 Sam Herceg: *Introduction to the relationship between Aliasing and Analog Filtering*, Herceg Engineering, 2008.
- 25 Microcomputer systems A/D- and D/A converters, Sven Knutsson, Chalmers Department of computer science and engineering, 2010
- 26 Elektriska mätsystem och mätmetoder, Lars Bengtsson, Pozkal, Poland 2008
- 27 Chopper- Stabilized Op Amps, Electronic design, 2009, <http://electronicdesign.com/article/analog-and-mixed-signal/chopper-stabilized-op-amps21449> (Acc. 2012-04-01)
- 28 EEVblog #24 Secret world of Chopper Amplifiers, David. L. Jones, 2009, <http://www.eevblog.com/2009/08/13/eevblog-24-the-secret-world-of-chopper-amplifiers/> (Acc. 2012-04-01)
- 29 Keith Pazul: *Controller Area Network (CAN) Basics*, Microchip Technology Inc, 1999.
- 30 Microcomputer systems Examples of communication interfaces, Chalmers Department of computer science and engineering, 2011
- 31 Wikimedia Commons :http://commons.wikimedia.org/wiki/File:Rs232_oscilloscope_trace.svg
- 32 MPLAB IDE User's guide, Microchip, 2009, http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_User_Guide_51519c.pdf, (Acc. 2012-05-01).
- 33 MPLAB C compiler for PIC 24 MCUs, Microchip, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en535364, (Acc. 2012-05-01).
- 34 MPLAB IDE v8 , Microchip, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002, (Acc. 2012-05-01).
- 35 PICkit3 Programmer/Debugger User's guide, Microchip, 2009, http://ww1.microchip.com/downloads/en/DeviceDoc/PICkit_3_User_Guide_51795A.pdf, (Acc. 2012-05-01)
- 36 *dsPIC30F3014/4013 Data Sheet*, Microchip, 2010, <http://ww1.microchip.com/downloads/en/DeviceDoc/70138G.pdf> (Acc 2012-04-28)
- 37 *Serial Quick Reference Guide*, National Instruments, 2004.
- 38 *DsPIC30F Family referens Manual*, Microchip, 2005, <http://ww1.microchip.com/downloads/en/DeviceDoc/70046E.pdf> (Acc 2012-04-28).
- 39 *dsPIC30F3014/4013 Data Sheet*, Microchip, 2010, <http://ww1.microchip.com/downloads/en/DeviceDoc/70138G.pdf> (Acc 2012-04-28).
- 40 Get free PCB Design Software with the EAGLE Freeware version, Cadsoft, 2011, <http://www.cadsoftusa.com/download-eagle/freeware/?language=en> (Acc. 2012-05-09)
- 41 Schematic. Layout. Autorouter. Three modules embedded in a single interface, Cadsoft, 2011,

<http://www.cadsoftusa.com/eagle-pcb-design-software/product-overview/> (Acc. 2012-05-10)

42 TRACE BENDS: Their impact on near fields(and EMC), Karthik Raj Guruchandran, 2010

43 *CAN - Protocols and definitions, Auml automation*, 2011, <http://projekt.auml.se/homeautomation:can> (Acc 2012-05-23)



Källkod

main.c

```

/*****
/*****

/*
5  * File:      main.c.
  * Program:    Battery Status Module Main Source Code.
  * Authors:    (C) 2012 Johan Persson, Anders Reinholdsson.
  * Created:    2012-03-19.
  * Updated:    2012-05-15.
10 */

/*****
/* Include-files */
/*****

15 /* Generic header file for the dsPIC30F4013. */
#include <p30f4013.h>
/* Math.h is only used for powf function in the Peukert calculation. */
#include <math.h>

20 #include "macros.h"
#include "can.h"
#include "eeprom.h"
#include "io.h"

25 /*****

/* Select XT oscillator mode 10 MHz => Fosc = 10 MHz => Fcy = 2.5 MHz. */
_FOSC(XT)

30 /*****

/* Variables with a prefix consisting of private is meant only to be accessed
in functions. Functions with a prefix consisting of public can be accessed
everywhere including main. */

35 /* Variable used for the delay function. */
volatile unsigned int global_delay_counter_ms = 0;

/* Global arrays and pointers to hold the temporary raw AD-values. */
40 volatile unsigned long int private_ad_voltage_sum = 0;
volatile unsigned long int private_ad_current_sum = 0;
volatile unsigned int private_ad_voltage_counter = 0;
volatile unsigned int public_ad_voltage_counter = 0;
volatile unsigned int private_ad_current_counter = 0;
45 volatile unsigned int public_ad_current_counter = 0;

/* Global arrays and pointers to hold the corrected values for voltage and
current each second 30 seconds back in time. The pointer points to the last
written value and must not be changed by any other function than the timer 3
50 interrupt routine. This data takes up about 122 bytes of memory. */
volatile unsigned int public_ad_voltages_s[BUFFER_S];
volatile unsigned char public_ad_voltage_s_pointer = 0;
volatile unsigned int public_ad_currents_s[BUFFER_S];
volatile unsigned char public_ad_current_s_pointer = 0;

55 /* Global arrays and pointers to hold the corrected values for voltage and
current each half minute 15 minutes back in time. The pointer points to the last
```

```

written value and must not be changed by any other function than the timer3
interrupt routine. This data takes up about 122 bytes of memory. */
60 volatile unsigned int public_ad_voltages_hm[BUFFER_HM];
   volatile unsigned char public_ad_voltage_hm_pointer = 0;
   volatile unsigned int public_ad_currents_hm[BUFFER_HM];
   volatile unsigned char public_ad_current_hm_pointer = 0;

65 /* Variable to hold amount of transfered charge out of the battery. The unit
   used is 1 As = 1 C and the format is 24.8-bit fixed point. This means that the
   maximum countable charge is about 4660 Ah. */
   volatile unsigned long int private_charge_out = 0;

70 /* Variable used to count seconds to a half minute. */
   volatile unsigned char private_timer3_seconds = 0;

   /* Variable used to count number of seconds the system has been idle. */
   volatile unsigned int public_idle_time_s = 0;

75 /* Variable to hold State-of-Charge in percent. */
   int public_soc_frac = 0;

   /* Variable to hold State-of-Health in percent. */
80 int public_soh_frac = 0;

   /* Variable to hold the average runtime current. */
   int public_i_avg_a_frac = 0;

85 /* Variable to hold the battery's real capacity. */
   unsigned int public_q_tot = 0;

   /* Variable to hold the battery's available capacity. */
   unsigned int public_q_ava = 0;

90 /* Variable to hold the remaining runtime in seconds. */
   unsigned int public_tr_s = 0;

   /* Global state type for use with the system state machine. */
95 enum state_type { start, transition, equilibrium, discharge };

   /* Variable to hold the calculated CPU-usage in percent. */
   volatile int public_cpu_usage_percent_frac = 0;

100 /* Variable to hold the CPU-calculation time in clock ticks. */
   volatile int public_cpu_calculation_time = 0;

   /* Variable to hold the information code from the system. */
   int public_info = 0;

105 /* Extern declaration of the struct for logging data to the EEPROM. */
   extern struct buffer r_w;

   /* Extern declaration of the struct for storing settings to the EEPROM. */
110 extern struct system_status;

   /* Variable used for testing the frequency of interrupt functions. */
   int toggle = 0;

115 /* Flag for controlling whether a write to the EEPROM is allowed. This is used
   to disable writes when the user is in the menu. */
   char eeprom_log_values = disable;

   /******

120 /* Interrupt routines. */
   void __attribute__((__interrupt__)) _T2Interrupt(void);
   void __attribute__((__interrupt__)) _T3Interrupt(void);
   void __attribute__((__interrupt__)) _ADCInterrupt(void);

125

```

```

/* Function for initialiazing the systems modules. */
void initialize(void);

/* Function to delay execution in functions. This can not however be used in
130 interrupt functions due to the delay variable being updated there. */
void delay(unsigned int time_ms);

/* Function for converting the raw ADC value to a fractional format voltage. */
int ad_to_voltage_v(unsigned int ad_value);
135 /* Function for converting the raw ADC value to a fractional format current. */
int ad_to_current_a(unsigned int ad_value);

/* Function to do a linear correct of a fractional number. The is used to in
140 conjunction with the calibration data and the voltage and current to do a linear
correction. */
int linear_correct(int value_frac, long int k_long_frac, long int m_long_frac);

/* Functions for retrieving voltage in time_s seconds back in time. The unit is
145 Volt and the format is a fractional number. */
int voltage_v_s(unsigned char time_s);

/* Functions for retrieving voltage in time_hm half-minutes back in time. The
unit is Volt and the format is a fractional number. */
150 int voltage_v_hm(unsigned char time_hm);

/* Functions for retrieving current in time_s seconds back in time. The unit is
Ampere and the format is a fractional number. */
int current_a_s(unsigned char time_s);
155 /* Functions for retrieving current in time_hm half-minutes back in time. The
unit is Ampere and the format is a fractional number. */
int current_a_hm(unsigned char time_hm);

160 /* Function for determening if the system has been idle in the set time. */
int is_idle(void);

/* Function to reset the counted transfered charge counter. */
void reset_charge_out(void);
165 /* Function for retrieving the transfered charge. The maximum returned counted
charge is about 18.2 Ah. The returned charge is in a 16 bit unsigned integer
form with a unit of 1 As = 1 C. */
unsigned int get_charge_out(void);
170 /* Function for converting EMF-voltage to State-of-Charge in percent. The
implementaion is done for several chemistries. */
int emf_to_soc_frac(int emf_frac);

175 /* Function for calculating the State-of-Charge based on the available and
transfered charge, in percent. */
int charge_to_soc_frac(unsigned int q_out);

/* Function for calculating the EMF-voltage in Volt as a fractional number using
180 the asympotes method. */
int asympotes_method(int sample_1_0m_frac, int sample_6_6m_frac);

/* Function for calculating the average current in Amperes as a fractional
number for the last x minutes. */
185 int average_current_frac(char minutes);

/* Function for updating the remaining run time in seconds. */
unsigned int time_remaining_s(int average_current_frac, unsigned int q_out);

190 /* Function for sending information about the system over RS-232. */
void send_rs_232(enum state_type state);

/* Function for sending information parametes over CAN. Observe that this must

```

```

    be disabled if the node is not connected to a CAN-bus. */
195 void send_can(void);

    /* Configuration menu with a timeout. */
    void menu(void);

200 /* Function to multiply a fixed point integer with an integer. */
    unsigned int multiply_frac_integer(unsigned int frac, unsigned int integer);

    /* Function for calculating the quotient between two integers. Returns percent in
    fractional format. */
205 int quotient_frac(unsigned int integer1, unsigned int integer2);

    /* Function for updating the State-of-Health. */
    unsigned int calculate_q_tot(int soc_start_frac, int soc_end_frac, unsigned int q_out);

210 /* Function for calculating the positive difference between two unsigned
    integers. */
    unsigned int uabs(unsigned int integer1, unsigned int integer2);

    /* Function for calculating the compensated current in Ampere as a fractional
    number according to Peukert's Law. */
215 int peukert_frac(int current_frac);

    /* Function for updating the system information code. */
    char update_info(int soc_frac, int soh_frac);

220 /*****

    int main(void)
    {
225         enum state_type state = start;
        int soc_start_frac;

        initialize();

230         print("\n\nSystem started and initialized.\n\n");

        /* Read the stored settings from the EEPROM. */
        eeprom_read_status_struct();

235         /* Present the configuration menu to the user. If the user does not respond
        the menu times out and exits after 5 seconds. */
        menu();

        /* Writes to the EEPROM is now allowed for the interrupt routine each 30
240 seconds. */
        eeprom_log_values = false;

        /* The main system loop. */
        while(true){
245             /* Always update remaining runtime. */
            public_tr_s = time_remaining_s(public_i_avg_a_frac, get_charge_out());

            /* Always update SoH = Qtot / Qnom. */
            public_soh_frac = quotient_frac(public_q_tot, status.q_nom);

250             /* State machine for the system algorithm. */
            switch(state){
                case start:
                    /* Update the start SoC and SoC via the EMF to SoC method. */
255                    soc_start_frac = public_soc_frac = emf_to_soc_frac(voltage_v_s(0));
                    /* Read Qtot from the EEPROM. */
                    public_q_tot = status.q_tot;
                    /* Read Iavg from the EEPROM. */
                    public_i_avg_a_frac = status.i_avg_a_frac;
260                    /* Update Qava = SoC * Qtot. */
                    public_q_ava = multiply_frac_integer(public_soc_frac, public_q_tot);

```

```

        break;
    case transition:
        /* Update the SoC via the Coulomb counting method. */
265     public_soc_frac = charge_to_soc_frac(get_charge_out());
        break;
    case equilibrium:
        /* Update the SoC using the asymptotes method. */
        public_soc_frac = emf_to_soc_frac(asymptotes_method(voltage_v_hm(11),
270 voltage_v_hm(0)));
        /* Check if the discharge is larger than the threshold. */
        if(get_charge_out() >= status.min_discharge_threshold_c){
            /* Check that the newly calculated capacity does not differ
            to much from the old capacity. */
275             if(uabs(calculate_q_tot(soc_start_frac, public_soc_frac,
get_charge_out()), public_q_tot) < status.max_capacity_diff_c){
                /* Update with the battery's real capacity with the new
                capacity. */
                public_q_tot = calculate_q_tot(soc_start_frac, public_soc_frac,
280 get_charge_out());
            }
            /* Reset the transfered charge because enough charge has
            been transfered for the algorithm to use it in calculations.
            */
285             reset_charge_out();
        }
        /* Update Qava = SoC * Qtot. */
        public_q_ava = multiply_frac_integer(public_soc_frac, public_q_tot);
        break;
290     case discharge:
        /* Update the SoC via the Coulomb counting method. */
        public_soc_frac = charge_to_soc_frac(get_charge_out());
        /* Update the average runtime current for the last 10 minutes
        only if the the average current the last 10 minutes is over the
295         current threshold. */
        if(average_current_frac(10) > status.idle_current_threshold_a){
            public_i_avg_a_frac = average_current_frac(10);
        }
        break;
300 }

/* Update the info parameter code to be sent over CAN. */
public_info = update_info(public_soc_frac, public_soh_frac);

305 /* Store the system parameters to the EEPROM buffer. */
if(eeprom_log_values){
    /*eeprom_store_system_info("a", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);*/
    eeprom_store_system_info(public_info, voltage_v_s(0), current_a_s(0),
    public_i_avg_a_frac, public_soc_frac, public_soh_frac, status.q_nom, public_q_tot,
310 public_q_ava, get_charge_out(), public_tr_s);
    eeprom_log_values = false;
}

/* Send information about the system via RS-232. */
315 send_rs_232(state);

/* Send information via CAN. Again this must be disabled if the node is
not connected to a CAN-bus. */
/*send_can();*/
320

/* Wait a couple of seconds so that the information is readable. */
delay(5000);

/* The next state part of the state machine. */
325 switch(state){
    case start:
        /* When in start state, always transition to the transition
        state. */
        state = transition;

```

```

330         break;
        case transition:
            if(is_idle()){
                /* If the system has been idle in the set time the system
                can transition to the equilibrium state. */
335                state = equilibrium;
            }else if(current_a_s(0) > status.idle_current_threshold_a){
                /* If the current is higher than the threshold the system
                must enter the discharge state. */
                state = discharge;
340            }
            break;
        case equilibrium:
            if(current_a_s(0) > status.idle_current_threshold_a){
                /* If the current is higher than the threshold the system
                must enter the discharge state. The start value of State-of-
345                Charge must also be saved for the algorithm. */
                soc_start_frac = public_soc_frac;
                state = discharge;
            }
            break;
        case discharge:
            if(current_a_s(0) < status.idle_current_threshold_a){
                /* If the current is lower than the threshold the system can
                enter the transition state. */
355                state = transition;
            }
            break;
        default:
            /* If an unknown state is entered, always go to the start state.
            */
360            state = start;
            break;
    }
}

365 /* Main never returns. */
return(0);
}

370 /*****

void menu(void)
{
    char timeout_s = MENU_TIMEOUT_S, enter_menu = false, choice = 0;
375    unsigned int value = 0;

    while(timeout_s--){
        print("Press enter in %d seconds to enter configuration menu...\n", timeout_s);
        if(check() == '\n'){
380            enter_menu = true;
            print("You choose to enter the menu.\n");
            break;
        }
        delay(1000);
385    }
    if(enter_menu){
        eeprom_read_status_struct();
        while(true){
            print_line_thick();
390            print("Battery Status Module, Johan Persson, Anders Reinholdsson (C) 2012,
XDIN AB.\n");
            print_line_thin();
            print("Configuration menu:\n");
            print_line_thin();
395            print("(1) Idle current threshold:..... %f3 Ampere.\n",
status.idle_current_threshold_a);
            print("(2) Min discharge threshold:.... %d Coulumb.\n",

```

```

status.min_discharge_threshold_c);
    print("(3) Max capacity difference:.... %d Coulumb.\n",
400 status.max_capacity_diff_c);
    print("(4) Battery chemistry:..... ");
    switch(status.battery_chemistry){
        case LICO2:
            print("Lithium-ion (LiCoO2).\n");
405         break;
        case SBSB_SBCA:
            print("Low maintenance lead-acid (SbSb/SbCa).\n");
            break;
        case CACA:
410         print("Maintenance free lead-acid (CaCa).\n");
            break;
    }
    print("(5) Number of battery cells:.... %d cells.\n", status.battery_cells);
    print("(6) Nominal battery capacity:... %d Coulumb.\n", status.q_nom);
415    print("(7) Real battery capacity:..... %d Coulumb.\n", status.q_tot);
    print("(8) Peukert coefficient:..... %f3.\n", (int)(256 *
status.battery_peukert));
    print("(9) EEPROM log time:..... %d Minutes.\n", status.log_time_m);
    print("(10) << FORMAT EEPROM >>\n");
420    print("(11) << DOWNLOAD STORED VALUES FROM EEPROM >>\n");
    print("(12) << SAVE CHANGES >>\n");
    print("(13) << EXIT >>\n");
    print_line_thin();

425    while(true){
        print("Enter your choice please: ");
        scan("%d", &choice);
        /*clear_stdin();*/
        if(1 <= choice && choice <= 13){
430         break;
        }else{
            print("Please enter a correct choice.\n\n");
        }
    }

435    switch(choice){
        case 1:
            while(true){
                print("The current idle current threshold is: %f3 Amperes.\n",
status.idle_current_threshold_a);
440                print("The value determines the threshold for when the system is
allowed to enter the\negulilibrium state and should be little more than the systems
standby-current.\n");
                print("Please enter a new idle current threshold in 256 * x
Amperes: ");
445                scan("%d", &value);
                if(0 < value && value < DEC_TO_FRAC(10)){
                    break;
                }
                print("Please enter a current between 0 and 256 * 10
450 Amperes.\n");
            }
            status.idle_current_threshold_a = value;
            break;
        case 2:
455            while(true){
                print("The current min discharge threshold is %d Coulumb.\n");
                print("The value determines the minimum discharge that is valid
for updateing the SoH\nand should be at least 10 % of the battery capacity\n");
                print("Please enter a new min discharge threshold in x Coulumb:
460 ");
                scan("%d", &value);
                if(0 < value && value < (0.9 * status.q_nom)){
                    break;
                }
465                print("Please enter a min discharge threshold between 0 and %d

```

```

Coulumb.\n", 0.9 * status.q_nom);
    }
    status.min_discharge_threshold_c = value;
    break;
470 case 3:
    while(true){
        print("The current max capacity difference is %d Coulumb.\n",
status.max_capacity_diff_c);
        print("The value determines the max difference in capacity that
475 is allowed for\nupdating the capacity. Larger values will result in the capacity being
updated\nmore often but can result in a worse estimation of the capacity.\n");
        print("Please enter max capacity difference in x Coulumb: ");
        scan("%d", &value);
        if(0 <= value && value < (0.5 * status.q_nom)){
480 break;
        }
        print("Please enter a min discharge threshold between 0 and %d
Coulumb.\n", 0.5 * status.max_capacity_diff_c);
    }
485 status.max_capacity_diff_c = value;
    break;
case 4:
    print("The current battery chemistry is ");
    switch(status.battery_chemistry){
490 case LICO02:
        print("Lithium-ion (LiCoO2).\n");
        break;
        case SBSB_SBCA:
            print("Low maintenance lead-acid (SbSb/SbCa).\n");
495 break;
            case CACA:
                print("Maintenance free lead-acid (CaCa).\n");
                break;
    }
500 while(true){
        print("What battery chemistry do you want to use?\n");
        print("(1) Lithium-ion (LiCoO2).\n");
        print("(2) Low maintenance lead-acid (SbSb/SbCa).\n");
        print("(3) Maintenance free lead-acid (CaCa).\n");
505 print("Please enter your choice: ");
        scan("%d", &value);
        if(1 <= value && value <= 3){
            break;
        }
510 print("Please enter a correct choice.\n");
    }
    status.battery_chemistry = value;
    break;
case 5:
515 while(true){
        print("The current number of battery cells is %d cells.\n",
status.battery_cells);
        print("Please enter number of battery cells in x cells: ");
        scan("%d", &value);
520 if(1 <= value && value <= 12){
            break;
        }
        print("Please enter number of battery cells between 0 and
12.\n");
525 }
    status.battery_cells = value;
    break;
case 6:
    print("The current nominal battery capacity is %d Coulumb.\n",
530 status.q_nom);
    print("Plese enter a new battery capacity in x Coulumb: ");
    scan("%d", &value);
    /* The value need not to be checked because it can have all the

```

```

values of an unsigned int. */
535         status.q_nom = value;
           break;
           case 7:
               print("The current real battery capacity is %d Coulumb.\n",
status.q_tot);
540               print("Plese enter a new real battery capacity in x Coulumb: ");
               scan("%d", &value);
               /* The value need not to be checked because it can have all the
values of an unsigned int. */
               status.q_tot = value;
545               break;
           case 8:
               while(true){
                   print("The current Peukert coefficient is %f3.\n", (int) (256 *
status.battery_peukert));
550                   print("Please enter a new Peukert coefficient in 256 * x: ");
                   scan("%d", &value);
                   if(0 <= value && value <= 512){
                       break;
                   }
555                   print("Please enter a Peukert coefficient between 0 and 2 * 256 *
x.\n");
                   }
                   status.battery_peukert = value / 256;
                   break;
560           case 9:
               while(true){
                   print("The current EEPROM log time is %d Minutes.\n",
status.log_time_m);
565                   print("Please enter a new EEPROM log time in x Minutes: ");
                   scan("%d", &value);
                   if(0 <= value && value <= 360){
                       break;
                   }
570                   print("Please enter a EEPROM log time between 0 and 360
Minutes.\n");
                   }
                   status.log_time_m = value;
                   break;
           case 10:
575               print("Do you really want to format the EEPROM (y/n)? ");
               choice = get();
               /*clear_stdin();*/
               if(choice == 'y'){
                   eeprom_format();
580                   print("\nEEPROM formating finished.\n");
               }else{
                   print("The EEPROM was NOT formated.\n");
               }
               break;
585           case 11:
               eeprom_send_system_info_rs232();
               break;
           case 12:
590               eeprom_write_status_struct();
               print("The settings has been saved.\n");
               break;
           case 13:
               return;
       }
595     }
}

/*****/
600 void send_rs_232(enum state_type state)

```

```

{
    int number;

605    print_thick_line();
    switch(state){
        case start:
            print("State:..... Start.\n");
            break;
610        case transition:
            print("State:..... Transition.\n");
            break;
        case equilibrium:
            print("State:..... Equilibrium.\n");
615            break;
        case discharge:
            print("State:..... Discharge.\n");
            break;
    }
620    print_thin_line();
    print("Voltage:..... %f3 Volt.\n", voltage_v_s(0));
    print("Voltage S/s:..... %d.\n", public_ad_voltage_counter);
    print("Current..... %f3 Ampere.\n", current_a_s(0));
    print("Current S/s:..... %d.\n", public_ad_current_counter);
625    print("Runt. current last 10 min:.... %f2 Ampere.\n", average_current_frac(10));
    print("Peukert comp. current:..... %f2 Ampere.\n", peukert_frac(current_a_s(0)));
    print("State-of-Charge (SoC):..... %f2 %.\n", public_soc_frac);
    print("State-of-Health (SoH):..... %f2 %.\n", public_soh_frac);
    print("Nominal capacity (Qnom):..... %d Coulomb.\n", status.q_nom);
630    print("Real capacity (Qtot):..... %d Coulomb.\n", public_q_tot);
    print("Available capacity (Qava):.... %d Coulomb.\n", public_q_ava);
    print("Transferred charge (Qout):..... %d Coulomb.\n", get_charge_out());
    print("Remaining runtime (tr):..... %t.\n", public_tr_s);
    print("Idle-time:..... %t.\n", public_idle_time_s);
635    print("Status:\n");
    switch(public_info){
        default:
            print("..Unknown battery status.\n");
            break;
640        case 1:
            print("..The battery has reached it's end of life and should be
replaced.\n");
            break;
        case 2:
645            print("..The battery is very discharged (< 25 %).\n");
            break;
        case 3:
            print("..The battery is almost discharged (> 25 %).\n");
            break;
650        case 4:
            print("..The battery is partially charged (> 50 %).\n");
            break;
        case 5:
            print("..The battery is charged (> 75 %).\n");
655            break;
    }
    print_thin_line();
    print("CPU-usage:..... %f2 %.\n", public_cpu_usage_percent_frac);
    print("CPU-time:..... %d ticks.\n", public_cpu_calculation_time);
660    print("Next EEPROM address:..... %d.\n", status.eeprom_address);
    print_thin_line();
    print("The contents of the half-minute-based voltage array is:\n[");
    for(number = 0; number < BUFFER_HM / 2; number++){
        print("%f1 ", voltage_v_hm(number));
665    }
    print("\n ");
    while(number++ < BUFFER_HM){
        print("%f1 ", voltage_v_hm(number));
    }
}

```

```

670     print("]\n");
        print_thin_line();
        print("The contents of the half-minute-based current array is:\n[");
        for(number = 0; number < BUFFER_HM / 2; number++){
            print("%f1 ", current_a_hm(number));
675     }
        print("\n ");
        while(number++ < BUFFER_HM){
            print("%f1 ", current_a_hm(number));
        }
680     print("]");
    }

    /*****/

685 void send_can(void)
    {
        /* Send the information code over CAN with INFO_ID. */
        send_can_message((int)public_info, INFO_ID);

690     /* Send the current voltage over CAN with VOLTAGE_ID. */
        send_can_message(voltage_v_s(0), VOLTAGE_ID);

        /* Send the current current over CAN with CURRENT_ID. */
        send_can_message(current_a_s(0), CURRENT_ID);
695     /* Send the State-of-Charge over CAN with SOC_ID. */
        send_can_message(public_soc_frac, SOC_ID);

        /* Send the State-of-Health over CAN with SOH_ID. */
700     send_can_message(public_soh_frac, SOH_ID);

        /* Send the remaining runtime over CAN with TR_ID. */
        send_can_message(public_tr_s, TR_ID);
    }
705 /*****/

void initialize(void)
{
710     int counter = 0;

        /* * * Port Configuration * * */

        /* Set RB0 as output. */
715     TRISBbits.TRISB0 = OUTPUT;
        /* Set RB1 as output. */
        TRISBbits.TRISB1 = OUTPUT;
        TRISFbits.TRISF5 = OUTPUT;

720     /* * * UART Configuration * * */

        /* Set the baud rate. */
        U2BRG = 3;
725     /* Enable the UART2 module. */
        U2MODEbits.UARTEN = true;
        /* Enable transmission. */
        U2STAbits.UTXEN = true;

730     /* * * ADC Configuration * * */

        /* Set the A/D conversion clock. */
        ADCON3 |= 63;
        /* Start sampling after each conversion. */
735     ADCON1bits.ASAM = true;
        /* Alternate the sampling between MUXA and MUXB. */
        ADCON2bits.ALTS = true;

```

```

/* Set MUXA to An9 and MUXB to An10. */
ADCHS |= 0x0A09;
/* Generate an interrupt once every two samples. */
ADCON2 |= 0x0004;
/* Enable the A/D module. */
ADCON1bits.ADON = true;

745  /* * * I2C Configuration * * */

/* Workaround built in error. */
LATFbits.LATF2 = false;
/* Set the baud rate. */
750  I2CBRG = 22;
/* Enable the I2C-module. */
I2CONbits.I2CEN = true;

/* * * Timer 2 Configuration * * */
755  /* Set timer 2 period. */
PR2 = 2500;
/* Turn on timer2. */
T2CONbits.TON = true;

760  /* * * Timer 3 Configuration * * */

/* Set timer 3 prescaler to 1:64. */
T3CONbits.TCKPS1 = true;
765  T3CONbits.TCKPS0 = false;
/* Set timer 3 period. */
PR3 = 39063;
/* Turn on timer 2. */
T3CONbits.TON = true;

770  /* * * Timer 4 Configuration * * */

/* Set timer4 prescaler to 1:64. */
T4CONbits.TCKPS1 = true;
775  T4CONbits.TCKPS0 = false;

/* * * Timer 5 Configuration * * */

/* Set timer 5 prescaler to 1:64. */
780  T5CONbits.TCKPS1 = true;
T5CONbits.TCKPS0 = false;
/* Set timer 5 period. */
PR5 = 39063;
/* Turn on timer 5. */
785  T5CONbits.TON = true;

/* * * CAN Configuration * * */

/* Set CAN baud rate. */
790  C1CFG1 |= 1;
/* Request normal operation mode. */
C1CTRL = 0x0800;
/*C1CTRL = 0x0A00;*/
/* Wait for normal operation mode. */
795  print("Waiting for normal operation mode...\n");
while(C1CTRL & 0x00E0);
/*while(!(C1CTRL & 0x0040));*/

/* * * Variable initialization * * */
800  for(counter = 0; counter < BUFFER_S; counter++){
    public_ad_voltages_s[counter] = 0;
    public_ad_currents_s[counter] = 0;
}
805  for(counter = 0; counter < BUFFER_HM; counter++){

```

```

        public_ad_voltages_hm[counter] = 0;
        public_ad_currents_hm[counter] = 0;
    }

810    /* * * Interrupt Configuration * * */

        /* Disable interrupt nesting. */
        INTCON1bits.NSTDIS = true;
        /* Enable Timer2 interrupt. */
815    IEC0bits.T2IE = true;
        /* Enable Timer3 interrupt. */
        IEC0bits.T3IE = true;
        /* Enable Timer5 interrupt. */
        IEC1bits.T5IE = true;
820    /* Enable A/D interrupt. */
        IEC0bits.ADIE = true;
    }
    /*****

825 void __attribute__((__interrupt__)) _T2Interrupt(void)
    {
        START_MEASURE_CPU_USAGE;

        /* Increases the global millisecond counter one millisecond. */
830    global_delay_counter_ms++;

        /* Stops the sampling and starts the conversion. Sampling auto starts after
        conversion complete. */
        ADCON1bits.SAMP = false;

835    /* Pin toggle test to measure the interrupt frequency at RB0. */
        if(toggle){
            LATBbits.LATB0 = false;
            toggle = false;
840        }else{
            LATBbits.LATB0 = true;
            toggle = true;
        }

845    /* Clear Timer2 interrupt flag. */
        IFS0bits.T2IF = false;

        STOP_MEASURE_CPU_USAGE;
    }

850    /*****

void __attribute__((__interrupt__)) _T3Interrupt(void)
    {
855    int array_pointer = 0;
        unsigned long int temp_voltage = 0, temp_current = 0;

        START_MEASURE_CPU_USAGE;

860    /* Clear watchdog timer. This should be done at least once every 16.384
        second. */
        CLRWDT();

        /* Below follows AD-values signal processing. */

865    /* Convert the 20-bit values for the voltage to 16-bit ones. */
        temp_voltage = 16L * private_ad_voltage_sum / private_ad_voltage_counter;
        private_ad_voltage_sum = 0;
        /* Update the number of successful voltage samples. */
870    public_ad_voltage_counter = private_ad_voltage_counter;
        private_ad_voltage_counter = 0;

        /* Convert the 20-bit values for the current to 16-bit ones. */

```

```

temp_current = 16L * private_ad_current_sum / private_ad_current_counter;
875 private_ad_current_sum = 0;
/* Update the number of successful current samples. */
public_ad_current_counter = private_ad_current_counter;
private_ad_current_counter = 0;

880 /* Convert the raw 16-bit AD-values to corresponding 8.8-bit fixed point
format voltage and current values. */
temp_voltage = ad_to_voltage_v(temp_voltage);
temp_current = ad_to_current_a(temp_current);

885 /* Do a linear correction of the voltage and current. */
temp_voltage = linear_correct(temp_voltage, AD_VOLTAGE_K, AD_VOLTAGE_M);
temp_current = linear_correct(temp_current, AD_CURRENT_K, AD_CURRENT_M);

/* Limit the transfered charge to the maximum battery capacity. */
890 if(private_charge_out >> 8 < status.q_nom){
/* Update the counter for the charge. */
private_charge_out += peukert_frac(temp_current);
}

895 /* Add the voltage and current values to the public second-based arrays. */
public_ad_voltages_s[public_ad_voltage_s_pointer] = temp_voltage;
public_ad_voltage_s_pointer = (public_ad_voltage_s_pointer + 1) % BUFFER_S;
public_ad_currents_s[public_ad_current_s_pointer] = temp_current;
public_ad_current_s_pointer = (public_ad_current_s_pointer + 1) % BUFFER_S;

900 /* Count the number of seconds the system has been idle. */
if(temp_current < status.idle_current_threshold_a){
/* There is no problem is this variable overflows. */
public_idle_time_s++;
905 }else{
public_idle_time_s = 0;
}

/* Check if half a minute has passed. */
910 if(private_timer3_seconds == 30){
private_timer3_seconds = 0;
/* Sum together all 8.8 fixed point format voltages and currents each
half minute. */
temp_voltage = 0;
915 temp_current = 0;

for(array_pointer = 0; array_pointer < BUFFER_S; array_pointer++){
temp_voltage += public_ad_voltages_s[array_pointer];
temp_current += public_ad_currents_s[array_pointer];
920 }

/* Divide the sum to get the mean value. */
temp_voltage /= BUFFER_S;
temp_current /= BUFFER_S;

925 /* Add the voltage and current values to the public half-minute-based
arrays. */
public_ad_voltages_hm[public_ad_voltage_hm_pointer] = temp_voltage;
public_ad_voltage_hm_pointer = (public_ad_voltage_hm_pointer + 1) % BUFFER_HM;
930 public_ad_currents_hm[public_ad_current_hm_pointer] = temp_current;
public_ad_current_hm_pointer = (public_ad_current_hm_pointer + 1) % BUFFER_HM;

/* Trigger logging of values to the EEPROM. */
if(eeprom_log_values != disable){
935 eeprom_log_values = true;
}
}else{
private_timer3_seconds++;
}

940 /* Clear Timer3 interrupt flag. */

```

```

        IFS0bits.T3IF = false;

        STOP_MEASURE_CPU_USAGE;
945 }

/*****/

void __attribute__((__interrupt__)) _T5Interrupt(void)
950 {
    /* Calculate the CPU-usge of the interrupt routines. */
    public_cpu_usage_percent_frac = 240L * public_cpu_usage_percent_frac / 256 + 16L *
100 * TMR4 / PR5;
    public_cpu_calculation_time = TMR4;

955    /* Clear Timer4 value. */
    TMR4 = 0;

    /* Clear Timer5 interrupt flag. */
960    IFS1bits.T5IF = false;
}

/*****/

965 void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
    int temp_volt;
    int temp_current;

970    START_MEASURE_CPU_USAGE;

    /* Retrieve raw voltage and current values from the ADC buffer. */
    temp_volt = ADCBUF0;
    temp_current = ADCBUF1;

975    /* Sum together all 256 raw 12-bit AD-values each second. */
    private_ad_voltage_sum += temp_volt;
    private_ad_current_sum += temp_current;
    private_ad_voltage_counter++;
980    private_ad_current_counter++;

    /* Clear the ADC interrupt flag. */
    IFS0bits.ADIF = false;

985    STOP_MEASURE_CPU_USAGE;
}

/*****/

990 int ad_to_voltage_v(unsigned int ad_value)
{
    /* 32-bit variable to hold the voltage value. */
    unsigned long int voltage;

995    /* Convert the raw 16-bit AD-value to a 8.8-bit fixed point format voltage.
    The raw AD-value corresponds to 0 - 30 Volts. */
    voltage = 1000L * ad_value;

    return voltage / 8533;
1000 }

/*****/

int ad_to_current_a(unsigned int ad_value)
1005 {
    /* 32-bit variable to hold the current value. */
    unsigned long int current;

    /* Convert the raw 16-bit AD-value to a 8.8-bit fixed point format current.

```

```

1010     The raw AD-value corresponds to 0 - 100 Ampere. */
        current = 100L * ad_value;
        return current >> 8;
1015 }

/*****/

int linear_correct(int value_frac, long int k_long_frac, long int m_long_frac)
1020 {
    /* Apply the linear correction by multiplying value_long_frac with k and
    add m. The calculation is done with 64-bit data-type "long long int". This
    is so that no data truncation occurs when multiplying one 8.8-bit number
    with one 16.16-bit number. The resulting 24.24-bit number is then converted
1025 to a 16.16-bit number. At last m is added and the number is converted to a
    8.8-bit number. */

    long long int value;

1030     value = (((value_frac * k_long_frac) >> 8) + m_long_frac) >> 8;

    return value;
}

1035 /*****/

int voltage_v_s(unsigned char time_s)
{
    time_s = public_ad_voltage_s_pointer - time_s - 1;
1040     time_s = time_s >= BUFFER_S ? BUFFER_S - 1 : time_s;
    return public_ad_voltages_s[time_s];
}

/*****/

1045 int voltage_v_hm(unsigned char time_hm)
{
    time_hm = public_ad_voltage_hm_pointer - time_hm - 1;
    time_hm = time_hm >= BUFFER_HM ? BUFFER_HM - 1 : time_hm;
1050     return public_ad_voltages_hm[time_hm];
}

/*****/

1055 int current_a_s(unsigned char time_s)
{
    time_s = public_ad_current_s_pointer - time_s - 1;
    time_s = time_s >= BUFFER_S ? BUFFER_S - 1 : time_s;
1060     return public_ad_currents_s[time_s];
}

/*****/

int current_a_hm(unsigned char time_hm)
1065 {
    time_hm = public_ad_current_hm_pointer - time_hm - 1;
    time_hm = time_hm >= BUFFER_HM ? BUFFER_HM - 1 : time_hm;
    return public_ad_currents_hm[time_hm];
}

1070 /*****/

int is_idle(void)
{
1075     return public_idle_time_s > IDLE_TIME_THRESHOLD_S ? true : false;
}

```

```

/*****/
1080 void reset_charge_out(void)
{
    private_charge_out = 0;
}

1085 /*****/

unsigned int get_charge_out(void)
{
    /* Return the value of the charge out counter converted from 20.8 type fixed
1090    point to a 16-bit unsigned integer. */
    return private_charge_out >> 8;
}

/*****/
1095 int emf_to_soc_frac(int emf_frac)
{
    int soc_frac = 0;

1100    emf_frac /= status.battery_cells;

    switch(status.battery_chemistry){
        case LICO02:
            if(emf_frac < DEC_TO_FRAC(3.00)){
1105                soc_frac = 0;
            }
            else if(DEC_TO_FRAC(3.00) <= emf_frac && emf_frac <
                DEC_TO_FRAC(3.22)){
                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.00)) *
1110 DEC_TO_FRAC(0.5) /
                DEC_TO_FRAC(3.22 - 3.00);
            }
            else if(DEC_TO_FRAC(3.22) <= emf_frac && emf_frac <
                DEC_TO_FRAC(3.54)){
1115                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.22)) *
                DEC_TO_FRAC(3.5) /
                DEC_TO_FRAC(3.54 - 3.22) + DEC_TO_FRAC(0.5);
            }
            else if(DEC_TO_FRAC(3.54) <= emf_frac && emf_frac <
1120                DEC_TO_FRAC(3.68)){
                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.54)) *
                DEC_TO_FRAC(4) /
                DEC_TO_FRAC(3.68 - 3.54) + DEC_TO_FRAC(4);
            }
            else if(DEC_TO_FRAC(3.68) <= emf_frac && emf_frac <
                DEC_TO_FRAC(3.80)){
1125                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.68)) *
                DEC_TO_FRAC(31) /
                DEC_TO_FRAC(3.80 - 3.68) + DEC_TO_FRAC(8);
            }
            else if(DEC_TO_FRAC(3.80) <= emf_frac && emf_frac <
                DEC_TO_FRAC(3.88)){
1130                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.80)) *
                DEC_TO_FRAC(22) /
                DEC_TO_FRAC(3.88 - 3.80) + DEC_TO_FRAC(39);
            }
            else if(DEC_TO_FRAC(3.88) <= emf_frac && emf_frac <
                DEC_TO_FRAC(3.98)){
1135                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.88)) *
                DEC_TO_FRAC(11) /
                DEC_TO_FRAC(3.98 - 3.88) + DEC_TO_FRAC(61);
            }
            else if(DEC_TO_FRAC(3.98) <= emf_frac && emf_frac <
                DEC_TO_FRAC(4.02)){
1145                soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(3.98)) *

```

```

DEC_TO_FRAC(9) /
    DEC_TO_FRAC(4.02 - 3.98) + DEC_TO_FRAC(72);
}
else if(DEC_TO_FRAC(4.02) <= emf_frac && emf_frac <
1150 DEC_TO_FRAC(4.06)){
    soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(4.02)) *
DEC_TO_FRAC(3) /
    DEC_TO_FRAC(4.06 - 4.02) + DEC_TO_FRAC(81);
}
1155 else if(DEC_TO_FRAC(4.06) <= emf_frac && emf_frac <
    DEC_TO_FRAC(4.08)){
    soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(4.06)) *
DEC_TO_FRAC(6) /
    DEC_TO_FRAC(4.08 - 4.06) + DEC_TO_FRAC(84);
1160 }
    else if(DEC_TO_FRAC(4.08) <= emf_frac && emf_frac <
    DEC_TO_FRAC(4.20)){
    soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(4.08)) *
DEC_TO_FRAC(10) /
1165 DEC_TO_FRAC(4.20 - 4.08) + DEC_TO_FRAC(90);
    }else{
        soc_frac = DEC_TO_FRAC(100);
    }
    break;
1170 case SBSB_SBCA:
    if(emf_frac < DEC_TO_FRAC(1.9813)){
        soc_frac = 0;
    }
    else if(DEC_TO_FRAC(1.9813) <= emf_frac && emf_frac <
1175 DEC_TO_FRAC(2.0096)){
        soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(1.9813)) *
DEC_TO_FRAC(25) /
        DEC_TO_FRAC(2.0096 - 1.9813);
    }
    else if(DEC_TO_FRAC(2.0096) <= emf_frac && emf_frac <
1180 DEC_TO_FRAC(2.0396)){
        soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(2.0096)) *
DEC_TO_FRAC(25) /
        DEC_TO_FRAC(2.0396 - 2.0096) + DEC_TO_FRAC(25);
1185 }
    else if(DEC_TO_FRAC(2.0396) <= emf_frac && emf_frac <
    DEC_TO_FRAC(2.0746)){
        soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(2.0396)) *
DEC_TO_FRAC(25) /
1190 DEC_TO_FRAC(2.0746 - 2.0396) + DEC_TO_FRAC(50);
    }
    else if(DEC_TO_FRAC(2.0746) <= emf_frac && emf_frac <
    DEC_TO_FRAC(2.1080)){
        soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(2.0746)) *
1195 DEC_TO_FRAC(25) /
        DEC_TO_FRAC(2.1080 - 2.0746) + DEC_TO_FRAC(75);
    }
    else{
        soc_frac = DEC_TO_FRAC(100);
1200 }
    break;
case CACA:
    if(emf_frac < DEC_TO_FRAC(1.9663)){
        soc_frac = 0;
1205 }
    else if(DEC_TO_FRAC(1.9663) <= emf_frac && emf_frac <
    DEC_TO_FRAC(1.9996)){
        soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(1.9663)) *
DEC_TO_FRAC(25) /
1210 DEC_TO_FRAC(1.9996 - 1.9663);
    }
    else if(DEC_TO_FRAC(1.9996) <= emf_frac && emf_frac <
    DEC_TO_FRAC(2.0663)){

```

```

soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(1.9996)) *
1215 DEC_TO_FRAC(25) /
DEC_TO_FRAC(2.0663 - 1.9996) + DEC_TO_FRAC(25);
}
else if(DEC_TO_FRAC(2.0663) <= emf_frac && emf_frac <
DEC_TO_FRAC(2.0996)){
1220 soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(2.0663)) *
DEC_TO_FRAC(25) /
DEC_TO_FRAC(2.0996 - 2.0663) + DEC_TO_FRAC(50);
}
else if(DEC_TO_FRAC(2.0996) <= emf_frac && emf_frac <
1225 DEC_TO_FRAC(2.1330)){
soc_frac = (unsigned long int)(emf_frac - DEC_TO_FRAC(2.0996)) *
DEC_TO_FRAC(25) /
DEC_TO_FRAC(2.1330 - 2.0996) + DEC_TO_FRAC(75);
}
1230 else{
soc_frac = DEC_TO_FRAC(100);
}
break;
}
1235 return soc_frac;
}

/*****/
1240 int charge_to_soc_frac(unsigned int q_out)
{
if(q_out < public_q_ava){
return 100L * 256 * ((public_q_ava - q_out)) / public_q_tot;
1245 }else{
return 0;
}
}

1250 /*****/

int asymptotes_method(int sample_1_0m_frac, int sample_6_6m_frac)
{
/* Implementation of the method presented by V. Pop, H.J. Bergveld, D.
1255 Danilov, P.P.L. Regtion and P.H.L. Notten: Battery Management systems,
Accurate Stateof-Charge Indication for Battery-Powered Applications, 2008,
p. 71-72. */

return (sample_6_6m_frac << 1) - sample_1_0m_frac;
1260 }

/*****/

int average_current_frac(char minutes)
1265 {
char counter = 0;
long int average_frac = 0;

/* Check that the value minutes is within limits. */
1270 if(2 * minutes < BUFFER_HM){
while(counter < 2 * minutes){
average_frac += current_a_hm(counter++);
}
average_frac /= counter;
1275 }else{
average_frac = 0;
}

return average_frac;
1280 }

```

```

/*****/
unsigned int time_remaining_s(int average_current_frac, unsigned int q_out)
1285 {
    unsigned int tr_s;
    if(q_out < public_q_ava && average_current_frac != 0){
        tr_s = 256L * (public_q_ava - q_out) / average_current_frac;
    }else{
1290         tr_s = 0;
    }
    return tr_s;
}

1295 /*****/

void delay(unsigned int time_ms)
{
    global_delay_counter_ms = 0;
1300    do{
        /* By executing the no-operation (NOP) instruction while waiting for
        time to pass the mikrocontroller saves some power. */
        NOP();
        NOP();
1305        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
1310        NOP();
        NOP();
        NOP();
    }while(global_delay_counter_ms < time_ms);
}

1315 /*****/

unsigned int multiply_frac_integer(unsigned int frac, unsigned int integer)
{
1320     return (unsigned long int)frac * integer / 256 / 100;
}

/*****/

1325 int quotient_frac(unsigned int integer1, unsigned int integer2)
{
    long int quotient;

    quotient = 256L * 100 * integer1 / integer2;
1330     return quotient;
}

/*****/

1335 unsigned int calculate_q_tot(int soc_start_frac, int soc_end_frac, unsigned int q_out)
{
    return 256L * 100 * q_out / (soc_start_frac - soc_end_frac);
}

1340 /*****/

unsigned int uabs(unsigned int integer1, unsigned int integer2)
{
1345     return (long int)integer1 - (long int)integer2 > 0 ? integer1 - integer2 : integer2 -
integer1;
}

/*****/

```

```

1350 int peukert_frac(int current_frac)
    {
        return (int) (256 * powf(((float)current_frac) / 256, status.battery_peukert));
    }
1355 /*****

char update_info(int soc_frac, int soh_frac)
{
1360     if(soh_frac < DEC_TO_FRAC(BATTERY_END_OF_LIFE_SOH_PERCETNT)){
        return 1;
    }else if(soc_frac < DEC_TO_FRAC(25)){
        return 2;
    }else if(soc_frac < DEC_TO_FRAC(50)){
1365     return 3;
    }else if(soc_frac < DEC_TO_FRAC(75)){
        return 4;
    }else{
        return 5;
1370     }
}

/*****/
/*****/

```

macros.h

```
/*
*****
*****
*/

/*
5  * File:      macros.h.
  * Program:    Battery Status Module Macros Source Code.
  * Authors:    (C) 2012 Johan Persson, Anders Reinholdsson.
  * Created:    2012-04-27.
  * Updated:    2012-05-15.
10 */

/*
*****

/* Macro for converting a decimal number to a fractional integer. The conversion
15 is done at the compiler time. Observe that the parantheses are necessary. */
#define DEC_TO_FRAC(d)          (int)((d) * 256)

/* * * General macros. * * */

20 #define true                  1
   #define false                0
   #define disable              -1
   #define NULL                 0
   #define INPUT                 1 /* Default. */
25 #define OUTPUT                0
   #define DIGITAL               1
   #define ANALOG                0 /* Default. */
   #define CLRWDT()              __asm__ volatile ("clrwdt")
   #define NOP()                 __asm__ volatile ("nop")
30 #define START_MEASURE_CPU_USAGE (T4CONbits.TON = true)
   #define STOP_MEASURE_CPU_USAGE (T4CONbits.TON = false)
   #define MENU_TIMEOUT_S        5

/* * * Size of the buffers used for storing voltage and current. * * */
35 #define BUFFER_S               30
   #define BUFFER_HM             30

/* * * Algorithm parameters. * * */
40 #define IDLE_CURRENT_THRESHOLD_A DEC_TO_FRAC(1)
   #define IDLE_TIME_THRESHOLD_S  396
   #define MIN_DISCHARGE_THRESHOLD_C 6336
   #define MAX_CAPACITY_DIFF_C    240
45 /* * * ID's used for CAN Communication. * * */

   #define VOLTAGE_ID              (0x0000)
   #define CURRENT_ID              (0x0008)
50 #define IAVG_ID                 (0x0010)
   #define SOC_ID                  (0x0018)
   #define SOH_ID                  (0x0020)
   #define INFO_ID                 (0x0028)
   #define TR_ID                   (0x0030)
55 /* * * Calibration data. * * */

   #define AD_VOLTAGE_K             (1.002146 * 65536)
   #define AD_VOLTAGE_M             (0.030000 * 65536)
60 #define AD_CURRENT_K             (1.003009 * 65536)
   #define AD_CURRENT_M             (0.125000 * 65536)

/* * * Battery configuration. * * */
```

```

65 /* Li-ion battery - LiCoO2. Source: V. Pop, H.J. Bergveld, D. Danilov, P.P.L.
    Region and P.H.L. Notten: Battery Management systems, Accurate Stateof-Charge
    Indication for Battery-Powered Applications, 2008. */
#define LICOO2 (1)
/* Lead-acid battery - Standard or Low Maintenance. Source: Bill Darden: Car and
70 Deep Cycle Battery Frequently Asked Questions 2012, http://www.batteryfaq.org/
   (Acc 2012-04-16). */
#define SBSB_SBCA (2)
/* Lead-acid battery - Maintance Free. Source: Bill Darden: Car and Deep Cycle
    Battery Frequently Asked Questions 2012, http://www.batteryfaq.org/ (Acc
75 2012-04-16). */
#define CACA (3)
/* Select the battery chemistry for the battery status module algorithms. */
#define BATTERY_CHEMISTRY (LICOO2)
/* Select the number of cells in the battery. */
80 #define BATTERY_CELLS (3)
/* Configure the nominal battery capacity in Coulombs. */
#define BATTERY_CAPACITY (4000 * 3.6)
/* Define the Peukert derating factor. */
#define BATTERY_PEUKERT (1.02f)
85 /* Define at which percent the battery's life should be considered used. */
#define BATTERY_END_OF_LIFE_SOH_PERCETNT (50)

/*****/
/*****/

```

can.h

```

/*****
/*****

/*
5  * File:      can.h.
  * Program:   Battery Status Module CAN Source Code.
  * Authors:   (C) 2012 Anders Reinholdsson, Johan Persson.
  * Created:   2012-04-30.
  * Updated:   2012-05-15.
10 */

/*****

/* Generic header file for the dsPIC30F4013. */
15 #include    "p30f4013.h"
  #include    "macros.h"
  #include    "io.h"

/*****
20 void send_can_message(int message, int identifier);

/*****
/*****/
```

can.c

```
/*
*****
*****
*/

/*
5  * File:      can.c
  * Program:    Battery Status Module CAN Source Code.
  * Authors:    (C) 2012 Anders Reinholdsson, Johan Persson.
  * Created:    2012-04-30.
  * Updated:    2012-05-15.
10 */

/*
*****
*****
*/

#include    "can.h"

15 /*
*****
*****
*/

void send_can_message(int message, int identifier)
{
20     /* Sets the bits in the standard identifier register SID. */
    /*Sets the frame mode to extended. */
    C1TX0SID = 0x20FD;
    /* Sets the bits in the extended identifier register EID. */
    C1TX0EID = 0xE000 | identifier;
25     /* Sets the number of data bytes to 2. */
    C1TX0DLC = 0x0810;

    /* Loading transmit buffer with message to be sent. */
    C1TX0B1 = message;
30     /* Setting this bit initiate the transmit. */
    C1TX0CONbits.TXREQ = true;
    /* TXREQ autoresets when message successfully sent. */
    while(C1TX0CONbits.TXREQ){
        /* Printing message error to console. */
35         if(C1TX0CONbits.TXERR){
            print("\n\n<< ERROR WHILE TRYING TO SEND ON THE CAN BUS >>\n\n");
        }else if(C1TX0CONbits.TXLARB){
            print("\n\n<< ARBITRATION ERROR WHILE TRYING TO SEND ON THE BUS >>\n\n");
        }else{
40             print("\n\n<< Waiting for the can message to be sent. >>\n\n");
        }
    }
    print("CAN message was sucessfully sent. (%d, %d)\n", message, identifier);
}

45
/*
*****
*****
*/
```

eeeprom.h

```
/*
*****
*****
*/

/*
5  * File:      eeeprom.h.
  * Program:    Battery Status Module EEPROM Source Code.
  * Authors:    (C) 2012 Johan Persson.
  * Created:    2012-04-25.
  * Updated:    2012-05-15.
10 */

/*
*****

/* Generic header file for the dsPIC30F4013. */
15 #include    <p30f4013.h>
#include    "io.h"
#include    "macros.h"

/*
*****

20 #define START_ADDRESS          (0x0000 + sizeof(status))
#define STATUS_ADDRESS          (0x0000)
/* The maximum possible address is 16383. */
#define LOG_TIME_M              (10)
25 #define SEND                  (0xA0)
#define RECIEVE                 (0xA1)
#define MAX_MEMORY_STRUCT      (2 * status.log_time_m)

/*
*****

30 struct buffer{
    char info;
    int voltage;
    int current;
35    int iavg;
    int soc;
    int soh;
    int qnom;
    int qtot;
40    int qava;
    int qout;
    int tr;
};

45 struct system {
    struct buffer *eeeprom_address;
    int idle_current_threshold_a;
    unsigned int min_discharge_threshold_c;
    unsigned int max_capacity_diff_c;
50    int battery_chemistry;
    int battery_cells;
    unsigned int q_nom;
    float battery_peukert;
    unsigned int q_tot;
55    int i_avg_a_frac;
    int cycles;
    int log_time_m;
};

60 /*
*****

void eeeprom_format(void);
void eeeprom_write_byte(int address, char word);
```

```

    char eeprom_read_byte(int address);
65 void eeprom_write_buffer_struct(void);
    void eeprom_read_buffer_struct(void);
    void update_eeprom_address(void);
    void reset_eeprom(void);
    void eeprom_write_status_struct(void);
70 void eeprom_read_status_struct(void);
    void eeprom_send_system_info_rs232(void);
    void eeprom_store_system_info(char into,int voltage, int current,
    int iavg, int soc, int soh, int qnom, int qtot, int qava, int qout, int tr);
    int max_address(void);
75
    /*****
    /*****

```

eeeprom.c

```
/*
*****
*****
*/

/*
5  * File:      eeeprom.c.
  * Program:    Battery Status Module EEPROM Source Code.
  * Authors:    (C) 2012 Johan Persson.
  * Created:    2012-04-24.
  * Updated:    2012-05-15.
10 */

/*
*****

/* Include the header file. */
15 #include "eeeprom.h"

/*
*****

/* Definition of status and buffer struct. */
20 struct buffer r_w;
   struct system status;

/*
*****

25 void eeeprom_format(void)
{
    /* Sets next EEPROM address to the first buffer address. */
    reset_eeeprom();
    char percent = 0;
30    int i, max_value;

    /* Sets the last address to erase when formatting EEPROM. */
    max_value = max_address() + sizeof(r_w);

35    /* Check the limits of this for-statement. */
    for (i=0; i <= max_value; i++){
        /* Writing end of string to every byte in EEPROM within limits. */
        eeeprom_write_byte(i, '\0');
        /* Tracking reset process in percent- */
40        if(!(i % (max_value / 10))){
            print("%d%, ", 10 * percent++);
        }
    }

45    /* Setting first EEPROM address as next address. */
    status.eeeprom_address = START_ADDRESS;
    /* Writing default values to status struct. */
    status.idle_current_threshold_a = IDLE_CURRENT_THRESHOLD_A;
    status.min_discharge_threshold_c = MIN_DISCHARGE_THRESHOLD_C;
50    status.max_capacity_diff_c = MAX_CAPACITY_DIFF_C;
    status.battery_chemistry = BATTERY_CHEMISTRY;
    status.battery_cells = BATTERY_CELLS;
    status.q_nom = BATTERY_CAPACITY;
    status.battery_peukert = BATTERY_PEUKERT;
55    status.q_tot = BATTERY_CAPACITY;
    status.i_avg_a_frac = 0;
    status.cycles = 0;
    status.log_time_m = LOG_TIME_M;
    /* Writing status struct to EEPROM. */
60    eeeprom_write_status_struct();
}

/* Function for filling the r_w buffer and sending it to eeeprom. */
```

```

void eeprom_store_system_info(char info,int voltage, int current,
65 int iavg, int soc, int soh, int qnom, int qtot, int qava, int qout, int tr)
{
    status.q_tot = qtot;
    status.i_avg_a_frac = iavg;
    eeprom_write_status_struct();
70
    r_w.info = info;
    r_w.voltage = voltage;
    r_w.current = current;
    r_w.iavg = iavg;
75
    r_w.soc = soc;
    r_w.soh = soh;
    r_w.qnom = qnom;
    r_w.qtot = qtot;
    r_w.qava = qava;
80
    r_w.qout = qout;
    r_w.tr = tr;
    eeprom_write_buffer_struct();
    update_eeprom_address();
    eeprom_write_status_struct();
85 }

/* Function for filling the r_w buffer and sending it through rs-232. */
void eeprom_send_system_info_rs232(void)
{
90
    int counter;
    print("<< PLEASE START COPY TEXT >>\n");
    print("Time, Info, Volt, Current, Iavg, SoC, SoH, Qnom, Qtot, Qava, Qout, Tr\n");
    for(counter = 0; counter < MAX_MEMORY_STRUCT; counter++){
        eeprom_read_buffer_struct();
95
        print("%d, ", 30 * counter);
        print("%d, %f3, %f3, ", (int)r_w.info, r_w.voltage, r_w.current);
        print("%f3, %f3, %f3, ",r_w.iavg, r_w.soc, r_w.soh);
        print("%d, %d, %d, ",r_w.qnom, r_w.qtot, r_w.qava);
        print("%d, %d",r_w.qout, r_w.tr);
100
        print_newline();
        update_eeprom_address();
    }
    print("<< PLEASE STOP COPY TEXT HERE >>\n\n");
}

105
/* Function for writing byte to EEPROM. */
void eeprom_write_byte(int address, char data)
{
    char address_high, address_low;
110
    address_high = (char)(address >> 8);
    address_low = (char)address; /* Dividing the address into two bytes,
    address_high and address_low. */

    /* Sending start condition to EEPROM. */
115
    I2CCONbits.SEN = true;

    /* Wait for end of master start sequence. */
    while(I2CCONbits.SEN);

120
    /* Writing EEPROM external address and write bit to tranceive register. */
    I2CTRN = SEND;

    /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

125
    I2CTRN = address_high; /* Writing address_high to tranceive register. */

    /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

130
    /* Writing address_low to tranceive register. */

```

```

I2CTRN = address_low;

/* Wait for acknowledge from slave. */
135 while(I2CSTATbits.TRSTAT);

/* Put the data to be sent in the transmission register. */
I2CTRN = data;

140 /* Wait for acknowledge from slave. */
while(I2CSTATbits.TRSTAT);

/* Send stop condition to EEPROM. */
I2CCONbits.PEN = true;

145 /* Wait for the stop condition sequence to finish. */
while(I2CCONbits.PEN);

/* Wait the required write cycle time. */
150 delay(10);
}

/*****

155 /* Function for reading byte from EEPROM. */
char eeprom_read_byte(int address)
{
    char address_high, address_low;
    address_high = (char)(address >> 8);
160 address_low = (char)address; /* Dividing the address into two bytes,
    address_high and address_low. */

    /* Wait for the bus to be idle. */
    //while(!I2CSTATbits.P);

165 /* Sending start condition to EEPROM. */
    I2CCONbits.SEN = true;

    /* Wait for end of master start sequence. */
170 while(I2CCONbits.SEN);

    /* Writing EEPROM external address and write bit to tranceive register. */
    I2CTRN = SEND;

175 /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

    I2CTRN = address_high; /* Writing address_high to tranceive register. */

180 /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

    /* Writing address_low to tranceive register. */
    I2CTRN = address_low;

185 /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

    /* Initiate repeated start condition. */
190 I2CCONbits.RSEN = true;

    /* Wait for the repeated start condition to finish. */
    while(I2CCONbits.RSEN);

195 /* Writing EEPROM external address and read bit to tranceive register. */
    I2CTRN = RECIEVE;

    /* Wait for acknowledge from slave. */
    while(I2CSTATbits.TRSTAT);

```

```

200      /* Enable receive mode for the I2C-module. */
      I2CCONbits.RCEN = true;

      /* Wait for the receive sequence to finish. */
205      while(I2CCONbits.RCEN);

      /* Sending NACK in next ack. sequence. */
      I2CCONbits.ACKDT = true;

210      /* Start next ack. sequence. */
      I2CCONbits.ACKEN = true;

      /* Wait for the acknowledge sequence to finish. */
      while(I2CCONbits.ACKEN); /* This might not be necessary. */

215      /* Sending stop condition to EEPROM. */
      I2CCONbits.PEN = true;

      /* Wait for the stop condition sequence to finish. */
220      while(I2CCONbits.PEN);

      return I2CRCV;
    }

225  /*****

      /* Function for writing buffer struct to eeprom. */
      void eeprom_write_buffer_struct(void)
      {
230          char *word_pnt, *eeprom_word_pnt;
          int i;

          word_pnt = (char*)&r_w;
          eeprom_word_pnt = (char*)status.eeprom_address;
235          for(i=0; i < sizeof(r_w); i++){
              eeprom_write_byte((int)eeprom_word_pnt, *word_pnt);
              word_pnt++;
              eeprom_word_pnt++;
          }

240      }

      /*****

      /* Function for reading buffer struct from eeprom. */
245      void eeprom_read_buffer_struct(void)
      {
          char *word_pnt, *eeprom_word_pnt;
          int i;
          word_pnt = (char*)&r_w;
          eeprom_word_pnt = (char*)status.eeprom_address;
250          for(i=0; i<sizeof(r_w); i++){
              *word_pnt = eeprom_read_byte((int)eeprom_word_pnt);
              word_pnt++;
              eeprom_word_pnt++;
255          }
      }

      /*****

260      /* Funktion for writing status struct to eeprom. */
      void eeprom_write_status_struct(void)
      {
          char *word_pnt, *eeprom_word_pnt;
          int i;
265          word_pnt = (char*)&status;
          eeprom_word_pnt = (char*)STATUS_ADDRESS;
          for(i=0; i<sizeof(status); i++){

```

```

        eeprom_write_byte((int)eeprom_word_pnt,*word_pnt);
        word_pnt++;
270     eeprom_word_pnt++;
    }
}

/*****/
275 /* Function for reading status struct from eeprom. */
void eeprom_read_status_struct(void)
{
    char *word_pnt, *eeprom_word_pnt;
280     int i;
    word_pnt = (char*)&status;
    eeprom_word_pnt = (char*)STATUS_ADDRESS;
    for(i=0; i<sizeof(status); i++){
        *word_pnt = eeprom_read_byte((int)eeprom_word_pnt);
285         word_pnt++;
        eeprom_word_pnt++;
    }
}

290 /*****/

/* Updating the eeprom address by adding size of struct buffer. */
void update_eeprom_address(void)
{
295     status.eeprom_address = (int)(++status.eeprom_address) < max_address() ?
    status.eeprom_address : (struct buffer*) START_ADDRESS;
}

/*****/
300 void reset_eeprom(void)
{
    status.eeprom_address = (struct buffer*)START_ADDRESS;
}

305 /*****/

int max_address(void)
{
310     return (sizeof(status) + 2 * status.log_time_m * sizeof(r_w));
}

/*****/
/*****/

```

io.h

```
/*
5  * File:      io.h.
  * Program:    Battery Status Module IO Source Code.
  * Authors:    (C) 2012 Anders Reinholdsson.
  * Created:    2012-04-24.
10 * Updated:    2012-05-15.
  */

/* Stdarg.h is used for variable number of parameters to functions. */
#include <stdarg.h>

15 /* Generic header file for the dsPIC30F4013. */
#include <p30f4013.h>

#define true 1
20 #define false 0
#define NULL 0

/*
25 void put(char character);
   char get(void);
   char check(void);
   void scan(char *string, ...);
   void print(char *string, ...);
30 void print_integer(int number);
   void print_frac(int frac, char precision);
   void print_long_frac(long int frac, char precision);
   void print_time(unsigned int time_s);
   void print_line_thick(void);
35 void print_line_thin(void);
   void print_newline(void);

/*
/*
```

io.c

```
/*
5  * File:      io.c.
  * Program:   Battery Status Module IO Source Code.
  * Authors:   (C) 2012 Anders Reinholdsson.
  * Created:   2012-04-24.
10 * Updated:   2012-05-15.
  */

/*
15 #include    "io.h"
  */

void put(char character)
{
20     U2TXREG = (int)character;
    while(U2STAbits.UTXBF);
}

/*
25 char get(void)
{
    while(!U2STAbits.URXDA);
    return U2RXREG;
30 }

/*
char check(void)
35 {
    return U2RXREG;
}

/*
40 void scan(char *string, ...)
{
    void *pointer = NULL;
    char character = 0;
45     int number = 0;

    va_list arglist;
    va_start(arglist, string);

50     while(*string != '\0'){
        if(*string == '%' && *(string + 1) == 's'){
            string += 2;
            pointer = va_arg(arglist, char*);
            do{
55                 character = get();
                *((char*)pointer) = character;
                pointer++;
            }while(character != '\n');
            *(char*)(pointer - 1) = '\0';
60         }else if(*string == '%' && *(string + 1) == 'd'){
            string += 2;
            while(true){
                character = get();
```

```

        if(character == '\n' || character < '0' || '9' < character){
65             break;
        }
        number = 10 * number + character - '0';
    }
    pointer = va_arg(arglist, int*);
70     *(int*)pointer = number;
}
}

75 /*****

void print(char *string, ...)
{
    long int number;
80     char *pointer = NULL;
    char character = 0;

    va_list arglist;
    va_start(arglist, string);
85     while(*string != '\0'){
        if(*string == '%' && *(string + 1) == 'd'){
            number = va_arg(arglist, int);

90             print_integer(number);

            string += 2;
        }else if(*string == '%' && *(string + 1) == 'c'){
            character = va_arg(arglist, char);
95             put(character);

            string += 2;
        }else if(*string == '%' && *(string + 1) == 'f'){
100             number = va_arg(arglist, int);

            print_frac(number, *(string + 2) - '0');

            string += 3;
        }else if(*string == '%' && *(string + 1) == 'l'){
105             number = va_arg(arglist, long int);

            print_long_frac(number, *(string + 2) - '0');

            string += 3;
        }else if(*string == '%' && *(string + 1) == 's'){
            pointer = va_arg(arglist, char*);
            string += 2;
            while(true){
115                 character = *pointer;
                if(*pointer == '\0'){
                    break;
                }
                put(*pointer++);
120             }
        }else if(*string == '%' && *(string + 1) == 't'){
            number = va_arg(arglist, unsigned int);
            string += 2;
            print_time(number);
125         }else{
            put(*string++);
        }
    }
}

130 /*****/

```

```

void print_integer(int number)
{
135     unsigned char n = 0, buffer[5];

        if(number < 0){
            put('-');
            number = -number;
140     }

        while(n < 5){
            buffer[n++] = number % 10;
            number /= 10;
145         if(!number){
            break;
        }
    }

150     while(n--){
        put('0' + buffer[n]);
    }
}

155 /*****

void print_frac(int frac, char precision)
{
    unsigned char n = 0, decimals, buffer[3];
160
    if(precision > 3){
        precision = 3;
    }else if(precision < 0){
        precision = 0;
165     }

    if(frac < 0){
        put('-');
        frac = -frac;
170     }

    decimals = frac >> 8;

    while(n < 3){
175         buffer[n++] = decimals % 10;
        decimals /= 10;
        if(!decimals){
            break;
        }
180     }

    while(n--){
        put('0' + buffer[n]);
    }
185
    if(precision){
        put('.');

        for(n = 0; n < precision; n++){
190             frac &= 0x00ff;
            frac *= 10;
            put('0' + (frac >> 8));
        }
    }
195 }

    /*****/

void print_long_frac(long int frac, char precision)

```

```

200 {
    unsigned char n = 0, buffer[5];
    unsigned int decimals;

    if(precision > 5){
205         precision = 5;
    }else if(precision < 0){
        precision = 0;
    }

210     if(frac < 0){
        put('-');
        frac = -frac;
    }

215     decimals = frac >> 16;

    while(n < 5){
        buffer[n++] = decimals % 10;
        decimals /= 10;
220         if(!decimals){
            break;
        }
    }

225     while(n--){
        put('0' + buffer[n]);
    }

    if(precision){
230         put('.');

        for(n = 0; n < precision; n++){
            frac &= 0xFFFF;
            frac *= 10;
235             put('0' + (frac >> 16));
        }
    }
}

240 /*****/

void print_time(unsigned int time_s)
{
    unsigned char h, m;
245     h = time_s / 3600;
    m = (time_s - 3600 * h) / 60;

    print("%d Hours, %d Minutes", h, m);
250 }

/*****/

void print_line_thick(void)
255 {
    print("\n=====
=====\\n");
}

260 /*****/

void print_line_thin(void)
{
    print("-----
265 --\\n");
}

```

```

/*****
270 void print_newline(void)
    {
        print("\n");
    }
275 *****/
/*****/
```

Användningsinformation

För att använda batteriövervakningssystemet som utvecklat under detta examensarbete erfordras lite information om hur systemet fungerar.

Vid start eller reset av systemet ges användaren en möjlighet att starta konfigurationsmenyn via RS-232-kommunikationen. I menyn är det möjligt att konfigurera några av de parametrar som algoritmen använder sig av.

För att ansluta till RS-232-kommunikationen skall terminalprogramvaran som används konfigureras för en baud rate om 38,4 kb/s, 8 databitar, ingen paritet samt 1 stoppbit. Radslutstecken skall endast bestå av line feed (LF). Den fysiska kopplingen görs med en okorsad DE-9-hona till DE-9-hane-kabel från serieporten hos en PC till modulens serieport.

Med modulen ansluten och matningsspänningen tillslagen skrivs texten ”*System started and initialized*” ut. Detta betyder att konfigurationen av modulens alla periferienheter har lyckats samt givetvis att RS-232-kommunikationen fungerar. Användaren har då 5 sekunder på sig att skicka ett enter-tecken om konfigurationsmenyn skall startas. I annat fall startar systemets algoritm att övervaka batteriet. Nedan visas ett exempel av menyn:

```
System started and initialized.

Press enter in 4 seconds to enter configuration menu...
Press enter in 3 seconds to enter configuration menu...
Press enter in 2 seconds to enter configuration menu...

You choose to enter the menu.

=====
Battery Status Module, Johan Persson, Anders Reinholdsson (C) 2012, XDIN AB.
=====
Configuration menu:
-----
(1) Idle current threshold:..... 1.000 Ampere.
(2) Min discharge threshold:.... 5760 Coulumb.
(3) Max capacity difference:.... 1440 Coulumb.
(4) Battery chemistry:..... Lithium-ion (LiCoO2).
(5) Number of battery cells:.... 6 cells.
(6) Nominal battery capacity:... 28800 Coulumb.
(7) Real battery capacity:..... 25920 Coulumb.
(8) Peukert coefficient:..... 1.019.
(9) EEPROM log time:..... 180 Minutes.
(10) << FORMAT EEPROM >>
(11) << DOWNLOAD STORED VALUES FROM EEPROM >>
(12) << SAVE CHANGES >>
(13) << EXIT >>
-----
Enter your choice please:
```

I menyn är det möjligt att ställa in de olika parametrarna genom att först välja vilken parameter som skall ställas in, och därefter följa anvisningen i menyn för respektive parameter.

Ytterligare alternativ för att formatera EEPROM, ladda ner kör-data, samt att spara inställningar finns. Värt att notera är att formatering av EEPROM endast sker för den valda log-tiden.

Tidplan

Nedan presenteras den tidplan som utgjorde grunden för planeringen av examensarbetet:

		Arbv. 1					Arbv. 2					Arbv. 3					Arbv. 4					Arbv. 5					Arbv. 6					Arbv. 7					Arbv. 8					Arbv. 9					Arbv. 10				
		12 Mar – 16 Mar					19 Mar – 23 Mar					26 Mar – 30 Mar					2 Apr – 6 Apr					9 Apr – 13 Apr					16 Apr – 20 Apr					23 Apr – 27 Apr					30 Apr – 4 Maj					7 Maj – 12 Maj					14 Maj – 18 Maj				
Ref	Moment	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F	M	T	O	T	F					
1	Inläsning	1	2	3	4	5																																													
2	Inköp	-	-																																																
3	Mikrokontroller						1	2	3																																										
4	Mätsystem, Hårdvara									1	2	3	4																																						
5	Kretskort													1	2	3	4																																		
6	Algorithm														1	2				3	-																														
7	Mjukvara																			-	1	2	3	4	5	6	7																								
8	Test & ink. Matriell																									1	2	3	-	-																					
9	Inst. Mtrl., Test																												-	-																					
10	CAN																										1	2		3	4	5																			
11	Justering, Rapportskr.																																				1	2	-	-	-										
12	Test, Färdst. Rprt.																																																		

