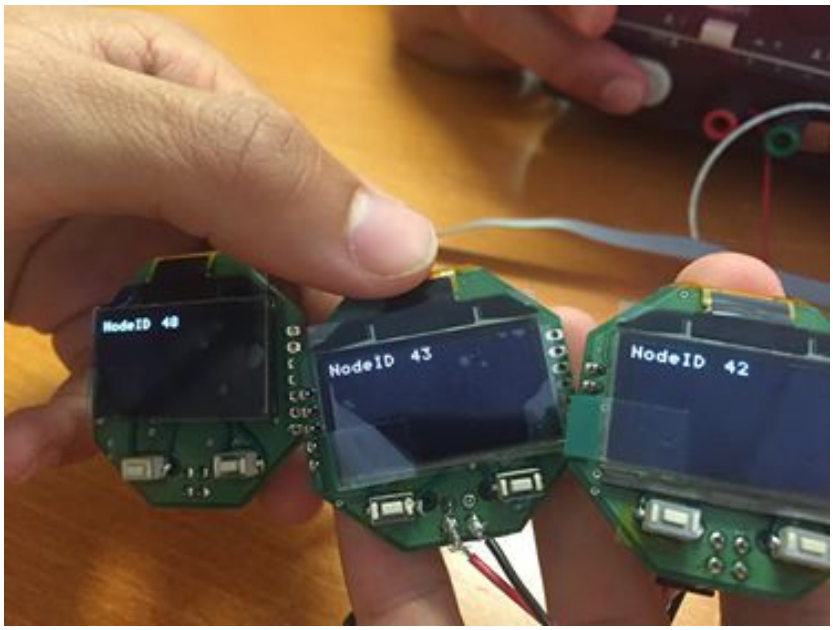


BMS med Dynamisk CANopen Nod-ID Tilldelning

Examensarbete inom Data- och Informationsteknik

JOHN EDWARDSON
EMIL EMANUELSSON

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

EXAMENSARBETE

BMS med Dynamisk CANopen Nod-ID Tilldelning

JOHN EDWARDSON
EMIL EMANUELSSON

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg 2018

BMS med Dynamisk CANopen Nod-ID Tilldelning

JOHN EDWARDSON
EMIL EMANUELSSON

Examinator: PETER LUNDIN

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag:
Testenheter som använts i projektet.

Institutionen för Data- och Informationsteknik Göteborg 2018

Sammanfattning

Detta projekt avser att fortsätta utvecklingen av en kontrollenhet för ett batteri i ett elfordon. Projektet har utförts för företaget Clean Motion AB i Chalmers lokaler. I syfte att kunna parallellkoppla flera batterier har det undersökts hur batteriernas kontrollenheter kan anslutas till en CAN buss. Detta skall konstrueras så att kontrollenheternas data ej kolliderar på CAN bussen. Kontrollenhetens mjukvara har även skrivits om från BASIC till C varefter dess funktioner har testas för att verifiera korrekthet.

I rapporten finns analys och beskrivning av batteristyrningssystemet (BMS) och dess delar som mjukvara berörs av. Information om hur CAN fungerar hämtas in för att få bättre förståelse för hur det applicerats i BMS. Därefter evalueras ett antal möjliga lösningar för att undvika kollision på CAN bussen mot varandra för att slutligen implementeras i systemet. Fungerande lösning har tagits fram och återfinns som två olika funktioner båda med olika fördelar.

Nyckelord: BMS, CAN, CANopen, I2C

Abstract

This project is intended to proceed with the developement of a battery management system (BMS) in an electric vehicle. The project has been issued from the company Clean Motion AB, and the work took place in a laboratory at Chalmers. In the purpose of being able to connect a set of batteries in parallel in the vehicle, the project examines the possibilities of connecting multiple BMS devices to a CAN bus. This should be done so that the data transmitted on the CAN bus does not collide with other data. Also the software of the BMS has been translated from BASIC to C and finally its features have been tested to verify the correctness.

In the report the reader will find an analysis and description of the BMS and its software. The reader will also find a description on how the CAN bus is operating and how the BMS uses it. Thereafter the report evaluates a set of different possible solutions for avoiding collision of data on the CAN bus. And how it was finally implemented and tested.

Keywords: BMS, CAN, CANopen, I2C

Förord

Detta examensarbete utfördes av John Edwardson och Emil Emanuelsson för företaget Clean Motion AB under vårterminen 2018. Examensarbetet omfattar 15 högskolepoäng och avslutar det tredje året på Elektroingenjörsprogrammet(180hp). Arbetet utfördes i Chalmers lokaler med Clean Motion AB som uppdragsgivare. Projektet tillkom på förfrågan till företaget som var positiva till att ta emot oss ex-jobbare. De gav ett antal förslag som sedan resulterade i ett projekt rörande deras batteristyrningssystem. Ett projekt där applikationen är mycket intressant och passar mycket väl in i utbildningen.

Vi tackar William Collings och Jesper Martaeng för all hjälp och att vi fick göra ex-jobbet på Clean Motion AB. Ett företag med en mycket intressant produkt, elfordonet Zbee.

Tack också till vår handledare på Chalmers, Jan Jonsson som ställt upp med guidning genom projektet.

Innehåll

Beteckningar & Ordlista

- **AFE:** Analog front-end. Ett konfigurerbart funktionsblock som också används som interface mellan olika sensorer och tex. en microkontroller.
- **BASIC:** Beginners All-purpose Symbolic Instruction Code. Ett tidigt programspråk känt för sin simplicitet.
- **BDU:** Battery Disconnect Unit.
- **BMS:** Battery Management System.
- **BMU:** Battery Management Unit.
- **Buss:** Menas databuss. En signalledning där många olika moduler skickar/tar emot data.
- **CAD:** Datorvektyg bl.a till för att designa elektriska kretsar.
- **CAN:** Controler Area Network.
- **EEPROM:** Electrically erasable programmable read-only memory.
- **LED:** Light Emitting Diode.
- **MCU:** Microcontroler Unit.
- **MOS-FET:** Fälteffekttransistor. Kan användas som strömbrytare.
- **OLED:** Organic Light-Emitting Diode. Används i skärmteknologi.
- **PCB:** Printed Circuit Board.
- **RAM:** Random Access Memory.
- **ROM:** Read-Only Memory.
- **SPI:** Serial Peripheral Interface.

1 Introduktion

1.1 Bakgrund

Företaget Clean Motion AB har utvecklat en liten, mycket energisnål, 3-hjulad mopedbil med syftet att ta fram en ny typ av hållbar transport. Clean Motion är ett ungt företag och deras produkt är mopedbilen Zbee. Med optimerad prestanda och låg vikt har de åstadkommit mycket låg driftsekonomi och energiförbrukning. Företaget arbetar hela tiden med att utveckla och förbättra Zbee. Hittills har man köpt en färdig lösning för bilens batteristyrningssystem (BMS). Detta gör det dock opraktiskt att tillämpa egna ändringar och lägga till eller ta bort funktioner från systemet. Därav finns intresse från företaget att designa en egen, anpassad BMS, med minimal funktion för att styra bilens batteri.

Clean Motion har påbörjat arbetet med att ta fram en egen lösning på batteristyrningen.

1.2 Syfte

Syftet med projektet är att fortsätta arbetet och hjälpa till med utvecklingen av BMS genom att byta programspråk från BASIC till C och addera ytterligare funktionalitet. Företaget vill designa en BMS som skall kunna möta en specifikation som är baserad på bl.a marknadsanalys och kundkrav.

1.3 Mål

Målet för projektet är att översatta befintlig programkod för BMS. Programmet skall översättas från BASIC till C och behålla samma funktion som tidigare. Ny kods funktion skall verifieras med tester. Dessutom skall en undersökning göras om hur man skulle kunna dynamiskt tilldela CAN-nod-identifierare för flera BMS i samma system vid systemstart till olika nod-ID .

1.4 Avgränsningar

Företaget är intresserat av att byta processor i BMS från typen PIC till STM som då även medför att man måste ta fram ett helt nytt kretskort vilket ej görs inom detta projekt. Koden skulle då även behöva anspassas till nya processorn mm.

2 Metod

Arbetetsuppgifter delas upp i nedan angivna rubriker. Arbetets utförande sker lämpligvis i den ordning som rubrikerna är angivna, och i viss mån så som arbetet måste struktureras upp eftersom delarna bygger på varandra.

2.1 Informationsinsamling

Informationsinsamling behandlar först att förstå och sätta sig in i hur BMS fungerar och är uppbyggd. Vilket dels är ett flertal olika funktioner i form av källkod och även hur den fysiska konstruktionen av kretskort med ingående komponenter, samt vilken del i systemet BMS utgör. Information skall även hämtas in om ingående teoretiska koncept (e.x CAN protokoll och SPI kommunikation) vilket ska finnas som grund till arbetsbeskrivningen.

2.2 Översättning av kod

Programkod skall översättas från programspråket BASIC till C. Språkens syntax differerar inte jättemycket. Detta innebär för det mesta en direkt översättning i syntax, dock finns en del funktioner i BASIC som ej har en direkt motsvarighet i C och behöver därför skrivas om.

2.3 Utökad funktion

När tidigare steg kan ses som färdiga påbörjas arbetet med att undersöka hur varje batteripack dynamiskt kan tilldelas en unik CAN nod-identifierare. Detta i syfte att man skall kunna parallellkoppla batterier. Ett ensamt batteri skickar i dagsläget ut sitt CAN nod-id på bussen vid uppstart. Adderas fler batterier uppstår en datakrock eftersom alla batterier har samma förinställda nod-id. En lösning skall tas fram på hur dessa skall ändras vid systemstart.

Uppgiften innehåller informationsinsamling om CAN, hur det är uppbyggt som nätverk och dess funktionsprinciper. Samt undersöka det nuvarande systemet för att komma fram till vilka möjligheter eller restriktioner som finns för att lösa problemet.

2.4 Testning

Systemets olika delar skall testas. Detta sker genom att simulera programkod samt presentera mätvärden mm. på en skärm.

2.5 Material & Utvecklingsverktyg

Ingående material för projektet bestod i två mindre versioner av BMU kretsen. Detta eftersom dessa var utrustade med kristaloscillatorer för bättre och stabilare klocka. De två mindre kretsarna är även utrustade med varsina monterad

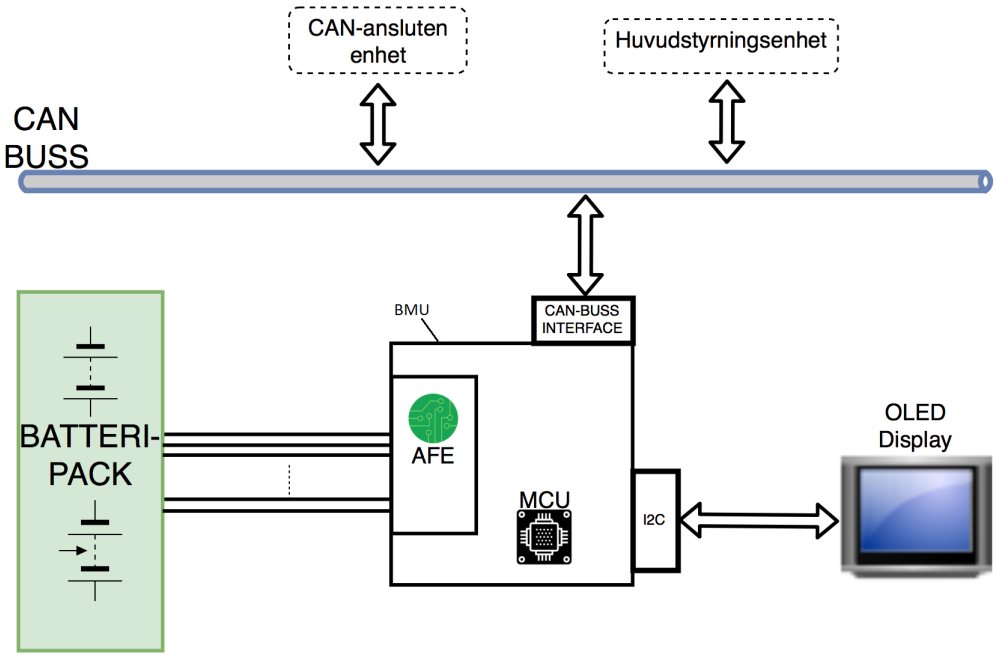
oled display vilket gör arbetet med CAN kommunikationen lättare då man direkt kan presentera data på skärmarna. För att programmera PIC processorerna på kretsarna användes PICkit 3 som programmerare och debug-verktyg.

För mjukvaruutvecklingen användes Microchips MPLAB-X IDE vilket är framtaget för PIC processorer. Kompilatorn XC8 som används tillsammans med MPLAB är också väldokumenterad vilket förenklar arbetet.

3 Teknisk bakgrund

3.1 Battery Management System Överblick

Batteristyrningssystemet är uppbyggt av olika moduler och innehåller bl.a BMU, BDU och CAN interface. Batteristyrningssystemets syfte är att skydda och övervaka batteriet och dess enskilda celler. När detta görs kommer man samtidigt maximera batteriets livslängd och säkerställa funktion i olika driftsförhållanden.



Figur 3.1: Modell över BMS, och dess anslutningar.

I figur 3.1 ses en övergripande bild av BMS. BMS är benämningen på hela systemet med anslutna delar medans BMU är den centrala enheten. På BMU finns en microkontroller samt en AFE-krets som är huvudkomponenterna på kretsen. På kretsen finns även ett CAN-buss interface och I2C anslutning. När flera batteri ska kopplas in i samma system så behöver varje batteri en egen BMU. Det är programvaran på MCUn på BMUn som innehåller ett nod-id som diskuteras senare i denna raport.

3.2 Microcontroler unit (MCU)

MCUn är centralenheten i BMS enheten och styr all funktion och kommunikation med övriga systemet. MCUn är av typen PIC (PIC18F25K80) och tillverkas av Microchip. Några egenskaper i MCUn som är intressanta för BMS:

- ECAN buss modul
- Låg sleep/run ström
- Snabb start-upp tid

3.3 Analog Front-end kretsen ML5238 (AFE)

På AFE kretsen finns nödvändig funktion för att övervaka batteriet. Den sköter all mätning av cellspänningar och styr balanseringen av batteriet. Balansering av ett batteri innebär att cellernas laddning hålls på en jämn nivå. På AFE finns även funktion för övervakning av upp- och urladdningsströmmar. Om en stor ström detekteras bryter kretsen automatiskt den MOS-FET som styr laddningen/urladdningen.

AFE kretsen kan styras utifrån inbyggda kontrollregister som i sin tur styrs från MCU via SPI interface. Genom att skriva till kontrollregistrerna ställer man exempelvis in vilken cellspänning eller ström man vill välja som utsignal till MCU.

3.4 Battery disconnect unit, BDU

BDU skall bryta/slå på strömmen till övriga systemet vid laddning/urladdning. Skall skydda batteriet från överanvändning.

3.5 Inter- Intergrated Circuit, I2C

I2C är en enkel tvåvägs-buss framtaget av Philips Semiconductors (nu NXP SC) för effektiv kommunikation mellan intergrerade kretsar. I2C används i projektet för kommunikation mellan MCU och en OLED display. Namnet I2C står för inter- intergrated circuit. Bussen består av två ledningar, en för seriell data och en för seriell klocka. I hemelektronik och andra inbyggda system återfinns ofta en liknande uppsättning komponenter. Nästan alla system innehåller någon form av microkontroller, minnesblock som RAM eller ROM, LED och LCD [1]. Dessa likheter mellan system låg som grund för framtagandet av I2C-bussen. Philips Semiconductors designade I2C för att gynna både utvecklare och tillverkare, då I2C gör utvecklingsarbetet effektivare och förenklar kretsdesign [1].

3.5.1 Utvecklingsfördelar

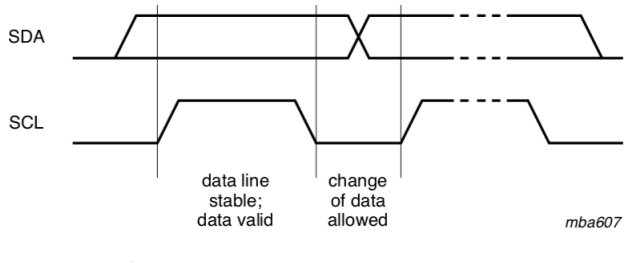
Kretsar med I2C kompatibilitet underlättar för utvecklare att designa kretsar eftersom det inte behövs något externt interface. Andra fördelar med I2C är integrerad adresserings och dataöverförings-protokoll vilket möjliggör för system att vara helt mjukvarubaserade. Fler fördelar är antalet anslutningar till I2C-bussen som finns för kommunicering, då det kan existera fler enheter på samma buss.

3.5.2 Tillverkarfördelar

I2C är inte bara fördelaktigt för utvecklaren men också tillverkaren. Att antalet ledningar som bussen behöver är få medför färre pins på integrerade kretsar, vilket i sin tur medför mindre och kostnadseffektivare kretsar. Eftersom I2C är helt integrerad tar det helt bort behovet av adresseringsavkodare och liknande som tar upp plats.

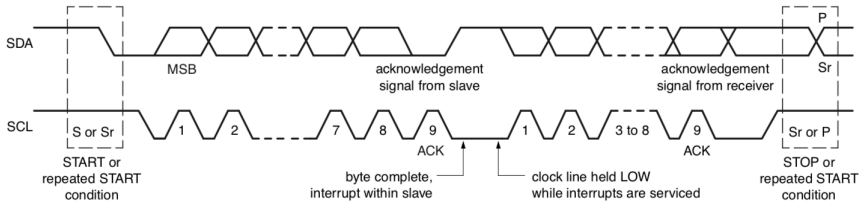
3.5.3 Data- och klocksignaler

Data- och klockledningarna är dubbelriktade ledningar och varje enhet som är ansluten till dessa har en unik adress. Varje enhet kan vara antingen sändare eller mottagare med undantag som t.ex LCD-drivare, som oftast bara är mottagare. En I2C-buss måste dock alltid ha en masterenhet, denna utgörs oftast av en microkontroller. Detta eftersom någon av enheterna behöver generera klocksignalen samt ställa in när en transfer skall starta och stoppa. Masterenheten kommer här att refereras till som sändare, och slave-enheten som mottagare. Varje transfer sker i paket med åtta bitar åt gången. Då är även timingen mellan enheter viktig för att överföringen skall fungera. Varje databit måste vara stabil undertiden klockpulsen är hög. Databiten ändras sedan endast under tiden klockpulsen är låg, se figur 3.2.



Figur 3.2: Klockning av en bitöverföring på I2C-bussen. *Hämtad: [1]*

För varje genererad klockpuls skickas en databit. Varje transmission börjar med en startbit och slutar med en stopbit. En övergång från hög till låg när klockpulsen är hög indikerar en startbit medans övergången låg till hög indikerar stopbit. Start och stop bitarna genereras alltid av masterenheten och bestämmer när bussen är upptagen. Efter startbiten följer datapaket om 8 bitar. Därefter måste en s.k Acknowledge-bit (ACK) skickas från mottagaren till sändaren för att bekräfta att byten togs emot korrekt och att nästa byte kan tas emot, se figur 3.3. När överföringen är klar avslutas överföringen med en stopbit.

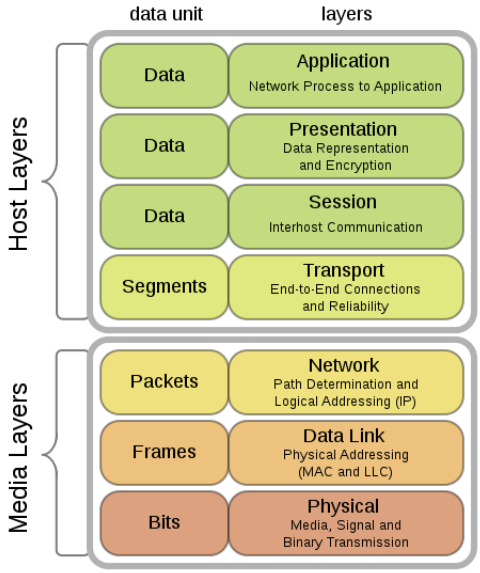


Figur 3.3: Figuren beskriver dataöverföringen på I2C-bussen. *Hämtad: [1]*

Om en mottagarenhet inte kan ta emot fler bytes innan den slutfört en annan funktion, t.ex en interruptfunktion, kan den sätta sändaren i väntetillstånd genom att hålla klocklänken låg. Överföringen fortsätter sedan när mottagaren är redo och frigör klocklänken, se figur 3.3. Ytterligare en signal finns för att indikera att en överföring inte gick bra, Not-ACK, vilken är användbar vid bl.a felsökning.

3.6 CAN nätverksprotokoll

Enligt ISO Open System Interconnection (OSI) sju-lager modell implementerar CAN främst de två lägre lagren i referensmodellen, se figur 3.4. OSI modellen är en standard som beskriver de olika funktionerna i ett kommunikationssystem och hur ett sådant nätverk är uppbyggt. OSI modellen är också ett sätt att enklare kunna beskriva komplexa nätverk genom att dela in nätverket i de olika lagren.



Figur 3.4: Referensmodell. *Hämtad: [2]*

Modellen i figur 3.4 hjälper till att beskriva och förstå nätverk som CAN. I efterkommande rubriker beskrivs lagren kort.

3.6.1 Fysiska lagret

Fysiska lagret utgör det fysiska medium som data skickas igenom, kan t.ex vara elektriskt eller optiskt. Detta lager behandlar även datakodning och anslutningar till hårdvara.

3.6.2 Datalänklagret

Datalänklagret implementerar överföringen mellan noder ovanpå det fysiska lagret. Varje nod har möjlighet att begära att få sända data vid alla tidpunkter. Om flera noder begär sändning samtidigt kommer den nod med högst prioritet att få tillgång till bussen.

3.6.3 Nätverkslagret

Nätverkslagret kontrollerar vilken väg data skall ta baserat på omständigheter i nätverket så som prioritet för ett meddelande.

3.6.4 Transportlagret

Transportlagret ser till att överföringen av data är säker och felfri. Transportlagret är också beroende av hur väl nätverkslagret fungerar. Ett opålitligt nätverkslager medför en mer utförlig feldetektering och korrektion i transportlagret.

3.6.5 Sessionslagret

Sessionslagret tillåter att två applikationsprocesser mellan två olika enheter att upprätta, använda och avsluta en förbindelse, kallat en session.

3.6.6 Presentationslagret

Presentationslagret formaterar data för att kunna presenteras till applikationslagret, liknat en översättare för nätverket. Lagret översätter data som används i applikationslagret till ett känt format för sändarenheten, och tvärtom vid mottagaren.

3.6.7 Applikationslagret

Applikationslagret är det lager som är närmast användaren. Det tillhandahåller funktioner för att konfigurera nätverket.

3.7 Controller Area Network, CAN

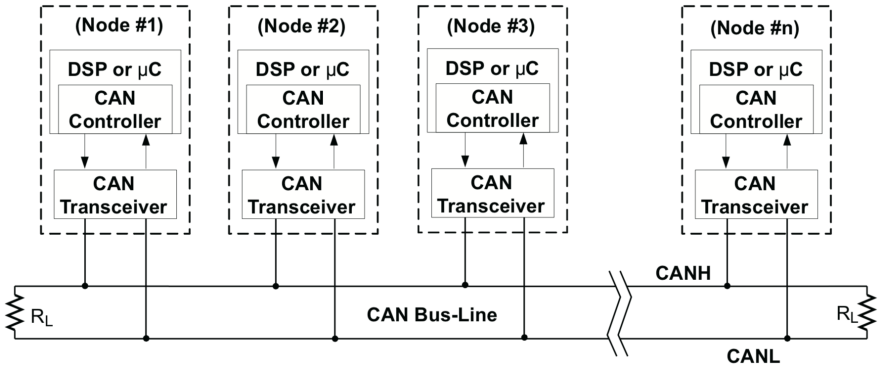
BMS skall anslutas till CAN buss. CAN är en metod för säker seriell kommunikation i olika applikationer. CAN var från början framtaget för att användas i fordon och har sedan blivit standardiserat och tillämpat inom fler marknader[3].

CAN är känt för att vara ett effektivt kommunikationsprotokoll anpassat för realtidssystem med hög säkerhet på informationsöverföringen. Det kan användas i fysiskt begränsade applikationer som kräver hög hastighet eller som behöver okomplicerade kopplingar mellan enheter. CAN möjliggör dessutom för produkter från olika tillverkare att enas om en standard för kommunikation i system. Det skiljer sig även från andra typer av nätverk då CAN inte nödvändigtvis skickar datapaket från en sändare till en specifik mottagare i ett nätverk. I CAN skickas istället datapaket ut till alla noder på nätverket, där noderna avgör om meddelandet på bussen skall läsas eller ignoreras. Detta bestäms av meddelandets identifierare som har information om vilken data meddelandet bär på.

3.7.1 CAN Funktionsprincip

Fördelarna med CAN är som sagt att man har en buss bestående av 2 ledningar och flera noder kopplat till dessa, därav slipper man kluster med kablage. CAN är även tåligt mot elektrisk störning och har felidentifiering som ger säkrare kommunikation.

CAN använder som tidigare beskrivet inte en specifik adress för att skicka data till en mottagare. Denna information finns istället i själva meddelandet [4]. Meddelandet består av olika delar eller så kallade ramar. En bitsekvens om elva bitar alt. 29 bitar innehåller en identifierare som berättar vilken typ av meddelande som skickas. Noderna på nätverket kan se denna identifierare välja att agera på- eller ignorera meddelandet.



Figur 3.5: CAN-bussens typiska uppbyggnad. *Hämtad:[5]*

3.7.2 CANopen

De övre fem lagren i OSI-modellen benämns ofta tillsammans som en typ av högnivå-protokoll. Det som skall användas i denna applikation och är det vanligaste kallas CANopen. CANopen har utvecklats till ett standardiserat inbyggt nätverk med stor konfigurationsflexibilitet. Denna standardisering underlättar för utvecklaren att hantera hårdvaruspecifika detaljer så som bit-timing och filtrering mellan enheter från olika tillverkare. CANopen täcker alltså de övre fem lagren: nätverks, transport, sessions, presentations och applikationslagret. Applikationslagret som ligger närmast användaren beskriver hur nätverket ska konfigureras, behandla datatransfer och synkronisering av andra CANopen enheter.

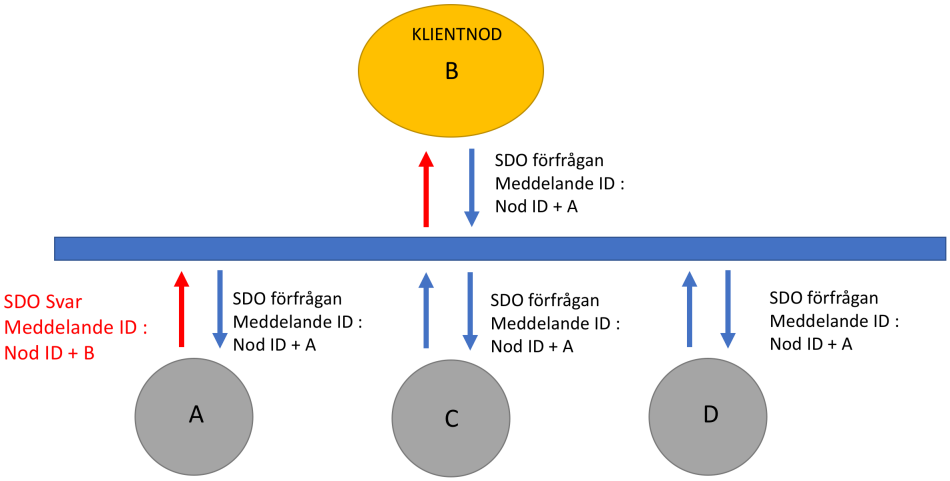
Alla CANopen-enheter har en objektlista (object dictionary, OD) där konfigurations- och processdata sparas. Objektlistan utgör på vilket sätt en CANopen enhet kan kommunicera. Man kan t.ex genom att styra tillverkarspecifika sektioner i obejktlistan trigga läsning eller skrivning av data. CANopens meddelandeformat är rambaserat vilket betyder att data överförs i paket. Varje paket innehåller

11 eller 29-bitars CAN-ID, kontrollbitar, datalängdsbita samt 0-8 byte data. CAN-ID i 11 bitar mode består av 4 bitar funktionskod samt 7 bitar nod-id. En 7-bitar lång nod-identifierare begränsar alltså antalet möjliga noder på samma buss till 127 st. På CANbussen finns en arbitrerare som skall se till att vid transmissionskollision mellan två meddelanden skall det meddelande med lägst identifierare få access till bussen först [8].

Två kommunikationssätt för att läsa obejktlistan hos en enhet kan ske med Service Data Objects (SDO) eller Process Data Objects (PDO). Enligt CANopen protokollet måste varje nod kunna hantera läs- och skrivning till dess objektlista. I projektet behandlas främst PDOer, men nedan följer även en kort beskrivning av SDOer för att lättare förstå kommunikationsprincipen.

3.7.3 Service Data Objects (SDO)

Tillvägagångssättet för att direkt läsa och skriva till objektslistan hos en nod sker med service data objects. Den nod som vars objektslista skall läsas kallas då server medan noden som hämtar datan kallas klient som också startar överföringen. Exempel på SDO kommunikation är att en nod sänder en dataförfrågan på nätverket innehållande ett meddelande-id. Noden som meddelandet är ämnat för svarar då med den efterfrågade datan. Även fast alla noder kan se meddelandet på nätverket är det bara noden med rätt nod-id som svarar. Datafältet i meddelandet innehåller ett index ämnat för vilket objekt som klientnoden vill läsa.



Figur 3.6: SDO exempel. Nod A ser meddelande-id och ser att meddelandet är en SDO förfrågan och svarar med efterfrågad data.

Denna typen av kommunikation används ofta då klientnoden i figur 3.6 fungerar som masternod. Lägg märke till skillnad mellan nod-id och meddelande-id.

3.7.4 Process Data Objects (PDO)

PDOer som använts i detta projekt representerar datainformation som ändras med tiden. Exempel på sådan typ av data är t.ex sensorer och styrsignaler som är anslutna till CAN-noden. Processdata återfinns också i objektlistan. Eftersom SDO endast kan ges access åt ett objekt åt gången, blir det opraktiskt att hämta mätdata som hela tiden ändras på detta sätt [6].

I CANopen finns därför kravet att en nod skall kunna skicka sin egen data på nätverket utan efterfrågan från en klient/masterenhet. Därför finns PDOer som delas in i två typer: TPDOer och RPDOer som står för Transfer- respektive Receive-PDO. Alltså TPDO för data som skickas från noden och RPDO för inkommande data till noden. PDO kan konfigureras till att fungera på olika sätt, bl.a med avseende på typen av PDO överföring. Några olika överföringssätt: händelsedriven, tidsdriven, individuell- eller synkroniserad överföring. Som namnet antyder kommer händelsedriven överföring triggas varje gång processdata ändras. Tidsdriven överföring sker med förutbestämda intervall vilket också används i detta projekt. Individuell överföring används mer sällan och gås ej igenom här. Synkroniserad överföring kan ske från t.ex en masternod som sickar ut en sync-signal på nätverket som en eller fler noder på nätet ser och kan svara på. Synkroniserad överföring är effektivt om en snabb överblick behövs över flera processer.

3.7.5 Network Management Protokoll, NMT

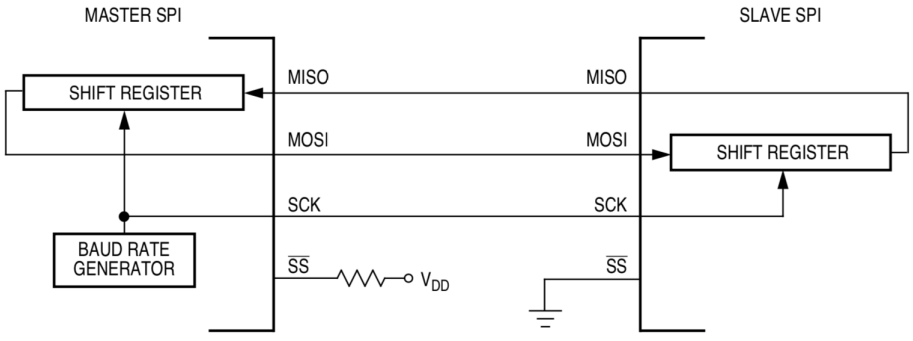
Network management definierar kommunikationstillståndet hos en CANopen enhet med en tillståndsmaskin. Tillståndsmaskinen kan befinna sig i initierings-, pre-operation-, operations- eller stoptillstånd, där det första tillståndet är initiering [6]. I projektet används endast delar av NMT protokollet. Detta eftersom initieringtillståndet skall användas för att bestämma nod-id samt att varje nod skall skicka ut s.k heartbeat-meddelanden. Heartbeat-meddelanden ska identifiera noder med felaktigt nod-id i systemet.

3.8 Serial Peripheral Interface, SPI

MCU är utrustad med Serial Peripheal Interface (SPI), vilket är en buss för synkron seriekommunikation med externa enheter. SPI kommunikation används i projektet mellan MCU och AFE-kretsen. Dock används inte det inbyggda interfacet i MCU p.g.a att man av misstag bytt plats på MCU-pinnarna MISO och MOSI i kretsschemat. Detta tar bort möjligheten att använda det inbyggda interfacet med nuvarande krets. Detta löses tillfälligt genom att använda s.k bit-banging för kommunikation mellan enheterna. Bit-banging är en metod som åstadkommer samma funktion i kretsen men kräver att en lösning i form av mjukvara tas fram istället. SPI består dock av följande signaler som skall efterliknas i mjukvaran hos MCU:

- CS - Chip- eller slavee select signal
- SCK - Serial clock
- MOSI - Master out slavee in
- MISO - Master in slavee out

SPI kommunikation mellan enheter använder master-slavee arkitektur, där masterenheten styr överföringen med klockan. I systemet är MCU master och AFE slavee. Det essentiella elementet i SPI systemet är dataregistren. Ett dataregister om 8-bitar i master-enheten och ett i slave-enheten utgör tillsammans ett 16-bitars register som är sammankopplat med MISO/MOSI pinnarna [11]. Vid en datatransfer skiftas 16-bitars registret 8 gånger med SCK från masterenheten, på detta sätt byter informationen plats mellan enheterna. Data som skrivs till master-enheten blir utsignal från slave-enheten och vise versa. Funktionsprincipen illustreras i figur 3.7.



Figur 3.7: Figuren beskriver förhållandet mellan Master/slavee. *Hämtad: [11]*

En justering mot figuren skall göras i projektet då SS signalen mellan master/slaveenheterna är sammankopplad. Detta eftersom det är vanligt att man använder

fler än en slave-enhet som styrs med SS. När AFE-kretsen tar emot data från MCU kommer MISO hos AFE ha hög impedans och därför inte skicka ut data. Denna funktion finns för att man först skall kunna välja vilket register man vill läsa/skriva på AFE-kretsen. Väljer man att skriva ett register gör sig AFE-kretsen redo för att ta emot 8 nya bitar, medans om man vill läsa ett register kommer AFE-kretsen skicka ut bitarna i det valda registret.

4 CANopen Dynamisk tilldelning av Nod-id

Den mer kreativa uppgiften i arbetet med BMS behandlar hur man kan åstadkomma parallellkoppling av batteriblock i fordonet. Det som förhindrar detta är att BMS:erna då har samma CAN nod-id. Det som skall undersökas är hur man få en automatisk funktion så att varje BMS får ett eget nod-id vid systemstart.

Idén är alltså att batterierna kopplas till CANbussen med samma funktion som tidigare. Dock har alla batterienheter samma nod-id vid start vilket orsakar en datakrock på CANbussen. Därför behövs en funktion som automatiskt ser till att alla enheter får ett eget nod-id vid start. Man vill t.ex inte behöva ändra i mjukvaran varje gång man installerar en enhet. Det är sedan tidigare känt att det ligger en viss svårighet i att lösa detta problem, men även ett antal färdiga lösningar [10]. Dock har detta projekt några restriktioner som gör att det inte direkt går att säga vilken implementation som är bäst.

Här vägs några olika möjliga lösningar mot varandra för att sedan dra slutsats om fördelar och nackdelar.

4.1 Nod-id med switchar

Vissa CANopen enheter använder switchar för att ställa in Nod-id. Detta kräver dock att man vid installering av enheten har kännedom om hur detta skall ställas in. Felmarginalen ökar för varje tillagd switch eftersom att det blir mer för montören att hålla koll på. Detta alternativ väljs bort i detta projekt eftersom det skall kunna implementeras i ett stort antal enheter samt att det i nuvarande lösning inte finns några switchar att använda. Denna metod används dessutom allt mindre över lag [10].

4.2 Nod-id med Nod-detektering

Denna metod tilldelar Nod-id direkt över nätverket vilket är en bra metod men kräver mycket av anslutna enheter. Detta förslag väljs tidigt bort eftersom det kräver att en masternod sköter tilldelning av Nod-id. I detta projekt skall alla noder tilldela sitt eget Nod-id och ingen masternod skall användas. Förbättringar i CANopen LSS (Layer Setting Services) talar dock för att denna metod skulle varit intressant att använda eftersom man har förbättrat och snabbat upp LSS Fastscan som söker av nätverket efter noder och tilldelar nod-id vid detektering [10]. Denna metod tillåter även plug-and-play så att en okonfigurerad nod kan läggas till när som helst och tilldelas nod-id.

4.3 Nod-id med fördröjningsfunktion

Till en början diskuterades hur man kunde implementera fördröjningar i from av en timer med olika längd som togs fram med hjälp av en slumpgenerator. Man skulle sedan tilldela nod-id utifrån de olika fördröjningarnas längd. Exempelvis

den som fått lägst fördröjning får lägst nod-id och den med längst fördröjning får högst nod-id. Svårigheten i detta ligger i att ta fram en slumpgenerator med ett unikt s.k frö, för varje enhet. Fröet som används i olika slumpgeneratorer är en utgångspunkt där slumpgenereringen startar ifrån. Skulle en slumpgenrerator utan frö användas finns en risk att två eller fler nod-id blir samma, vilket skulle orsaka en datakrock. Ett unikt frö är svårt att ta fram om det inte kan läsas in från en unik källa. Exempelvis skulle ett unikt enhets-id utgöra ett bra frö. Dock finns inget sådant unikt id att tillgå i PIC-processorn.

4.4 Tilldelning med Serienummer

Denna idé bygger på att använda batteriets serienummer för att tilldela nod-id. Batteriets serienummer består av elva bytes och finns att tillgå från minnet. Idén är att varje enhet skicka ut s.k heartbeat-meddelanden på CAN bussen vid start, alla har då samma nod-id. När enheterna läser ett meddelande på bussen med samma nod-id som sitt eget kommer detta trigga en funktion för utbyte av nod-id. Det nya nod-id kommer väljas utifrån den första siffran i det unika serienumret och sedan gå tillbaka till normal funktion. Skulle enheten dock läsa av ytterligare ett meddelande med samma nod-id, återupprepas tidigare funktion med den andra siffran i serienumret osv. Detta anses vara den bästa implementationen för tilldelning av nod-id då det går fortare än att använda fördröjningar. I ett system med olika enheter är detta ett bra sätt att fastställa nod-id tilldelning då det görs en gång vid utvecklingen av systemet[10]. När man sedan behöver byta ut enheter och göra systemunderhåll behöver man inte ta hänsyn till nod-id tilldelning. Det som dock begränsar denna idé i detta projektet är att formatet på batteriets serienummer ännu inte är definitivt.

4.5 Tilldelning med A/D omvandling

BMSens MCU har ett antal pinar som ska göra A/D omvandlingar på olika signaler. Dessa kan användas för att tilldela ett nytt nod-id om det är flera BMS i samma system då det är stor sannolikthet att signalerna till de olika MCUerna inte är exakt lika. Tanken är om en BMS upptäcker att det finns en annan enhet med samma nod-id på CAN bussen ska MCU göra en A/D omvandling på en analog ingång och sedan baserar det nya nod-id på resultatet.

4.6 Upptäcka dubbletter utav nod-id

Som tidigare beskrivet skickas meddelande id med 11 eller 29 bits med varje meddelande. I detta projekt används meddelande id med 11 bit. MCUn kan filtrera vilka meddelande den vill ta emot baserat på dessa meddelande id. MCUn kan ha flera filter för olika meddelande och den sparar vilket filter som tar emot. Detta kan användas för att upptäcka ifall det finns andra enheter med samma nod-id på samma CAN-buss då meddelande id innehåller nod-id. Om MCUn filtrerar efter en typ av meddelande som skickas ifrån någon med samma nod-id

som MCUn använder märker den då att det finns en annan enhet med samma nod-id ifall detta filter tar emot ett meddelande.

5 Genomförande

5.1 Översättning av Källkod

Översättning av källkod från BASIC till C görs med enkelhet sett till syntax då de liknar varandra. Stora delar av koden skrivs med tidigare kunskaper i C. I BASIC används subrutiner som är en typ av funktion. Subrutiner används när olika funktioner utför samma typ av operation. I C kallas subrutinerna endast för funktioner men används på samma sätt som i BASIC.

5.2 OLED och LCD Skärmar med I2C

I projektet används två olika skärmar. En OLED som används på den kompletta BMUn, denna ska visa bl.a. vilka celler som balanseras och hur mycket laddning som finns kvar i batteriet i procent. Den andra skärmen som använts i projektet är en LCD skärm som använts på testkretsen. Denna används till att visa vad som togs emot på CAN-bussen och vilket CAN nod-id som används.

Båda skärmarna kommunicerar med PICen genom en I2C buss. Dock är dom kopplade på olika sätt till PICen vilket innebär att dom inte har samma förutsättningar för I2C programmering på PICen. LCD skärmen är kopplad till de pins på PICen som har hårdvarustöd för I2C vilket gör den enklare att programera eftersom kompilatorn XC8 har färdiga C funktioner för användning utav hårdvarufunktionerna. Dessa funktioner går att hitta i dukumentet MPLAB XC8 Perifiral Libraries".

OLED skärmen är dock inte kopplad till pinar som har hårdvarustöd för I2C funktion vilket gör att de PIC anpassade funktionera för I2C inte går att använda, alltså måste I2C bitbangas, alltså styra klock och data signalerna med mjukvara istället för hårdvara. Eftersom det enbart är två noder på I2C bussen och enbart PICen som skickar till OLED skärmen behövs inte hela I2C protokollet utan bara funktioner för att skicka data. Eftersom det bara är en slav och en master kunde ack signalen också uteslutas ur funktionerna. Funktionerna för att skicka data byggdes upp utav mindre funktioner för att med enkelhet kunna utöka koden mer fler funktioner som kan skicka olika typer utav data. Grundfunktionen är en funktion som enbart skickar en data bit och en klocksignal på I2C bussen. Denna funktion används sedan för att bygga en funktion som skickar en byte. Med denna funktion kan de funktioner som anropas ifrån andra funktioner för att skicka till skärmen byggas. De funktionerna skickar tre eller fyra byte där första byten är adressen till skärmen, andra byten talar om för skärmen om det är data eller kommando som skickas och de sista byten är skickad data eller kommando.

5.3 Dynamisk nod-id tilldelning

När CAN fungerade kunde funktionen för att tilldela ett nytt nod-id testas. En metod behövs för att se ifall enheterna upptäcks och att de finns flera enheter med samma nod-id på CAN bussen.

För detta behövs det bestämmas på vilket sätt de ska upptäcka att det finns flera enheter med samma nod-id. Det bestämdes att ett puls meddelande ska skickas av BMS som varje enhet kan filtrera efter så som beskrivet i 5.6. Detta puls meddelande ska skickas varje sekund och innehålla information om BMSens status.

Sedan skapades två förslag på hur tilldelning utav ett nytt nod-id. Första förslaget baserades på batteriet serienummer och det andra på A/D omvandling.

6 Tester

6.1 Test utav översatta BMS funktioner

Då testkortet inte har någon input ifrån AFE och inte samma pin konfiguration kunde inte alla funktioner testas fullt ut. I en del funktioner testades det om A/D omvandlingarna fungerade och om värden kunde läsas in ifrån eeprom. Andra funktioner kunde testas enklare genom att använda simulerade inputs för att se om rätt värde från ett look-up table blir returnerat. De funktioner som hade uträkningar dubbelkollades att dom har samma uttryck som källkoden. Ett flertal funktioner använder en SPI funktion som inte gick att testa ordentligt då ingen SPI mottagare fanns tillgänglig. SPI testades enbart med ett oscilloskop där data och klock signalen undersöktes.

6.1.1 Sluttest

När den översatta koden testats så mycket som möjligt med testkorten kunde den testas på det kompletta BMS-kortet med en batterisimulator. Då kunde det säkerställas att SPI fungerade och att de inställningar som skiljde så som frekvensberoende parametrar och pin-tilldelning stämde.

SPI testades genom att presentera data på kortets skärm, där resultatet av vissa A/D mätningar visades. Detta visade sig fungera eftersom MCU skickar bl.a via SPI vilken cells spänning som skall läsas. Om överföringen inte blir korrekt kommer AFE inte att skicka någon signal medans om det fungerar kommer cellspänningen läggas ut för att A/D-omvandlas tillbaka till MCU.

6.2 Skicka och ta emot CAN

På ett av testkorten fanns ett BASIC program som hade testats för att ta emot CAN meddelanden och sedan skriva ut detta på LCD skärmen. För att istället

testa CAN funktionerna i C användes detta kort som mottagare. Om inte det skickade värdet skrevs ut på mottagar skärmen kontrollerades CAN kontrollregistren med hjälp av debuggverktyget. Det som orsakade problem var att baudrate inställningarna inte var anspassade. Då BMS använder en interoscillator måste denna skalas ned rätt för att enheterna skall vara synkroniserade.

När CAN funktionerna för att skicka data fungerade, skrevs BASIC programmet om till ett motsvarande C program. Detta program användes sen för att testa flera funktioner bl.a till att ta emot CAN meddelanden. När det fungerade att ta emot CAN meddelanden så modifierades programmets CAN mottagar funktioner till BMS huvudprogramet för att kunna ta emot enbart puls meddelandet.

6.2.1 Funktionstest CAN nod-id

Vid test av nod-id tilldelning ansluts två kretsar till CAN bussen och programmeras med funktionen som byter nod-id om nödvändigt och samma start nod-id. Sedan används displayerna för att skriva ut vilket nod-id som enheten har. Denna test har lyckats varje gång och inget fel har upptäckts.

6.2.2 Skärm test

Första målet för LCD skärmen var att se att I2C kommunikationen fungerade korrekt. Till detta användes enbart initieringsfunktionen och bakgrundsfunktionen i en while(1) loop. Signalerna mellan PICen och skärmen visades också på ett oscilloskop där frekvensen kontrollerades. I debuggprogrammet kontrollerades om skärmen svarade med en ACK signal.

När I2C kommunikationen fungerade kontrollerades alla utskriftsfunktioner. Det som kontrollerades var att alla tecken kunde skrivas och att dom aldrig skrev för nära varandra. Detta gjordes genom att skriva alla tecken och sedan flera tecken efter varandra.

Eftersom det enbart fanns tillgång till ett test kort utan OLED skärmen och utskriftsfunktionerna skiljer sig lite ifrån LCD kunde inte funktionerna för OLED testas fullständigt. Det som kunde testas var I2C bit-bangningen då båda skärmarna använder I2C. Detta gjordes genom att byta ut de hårdvarubaserade I2C funktionerna i LCD funktionerna till bit-bangnigs funktionerna. Även vid detta test koppades signalerna till ett oscilloskop för att studera signalerna.

7 Resultat

7.1 CAN nod-id tilldelning

Två förslag på hur ett unikt nod-id kan tilldelas till alla enheter i ett system med flera enheter som initialt har samma nod-id har tagits fram. Båda förslagen upptäcker att detta inträffat med hjälp utav ett puls-meddelande som också blivit utvecklat i samband med detta projekt. För detta har BMS ett CAN filter som tar emot meddelanden med TPDO för puls-meddelande med sitt eget nod-id. Puls-meddelandets funktion är baserat på CAN-opens NMT-meddelande och specifikationerna ifrån Clean Motion. Det som skiljer förslagen är på vilket sätt dom tilldelas nya nod-id. Båda förslagen uppdaterar också nod-id i eeprom för att inte behöva skapa ett nytt nod-id varje gång systemet startar. Genom att inte ändra nod-id varje gång systemet startar fås ett effektivare system, då tid inte behöver spenderas på nod-id tilldelning.

7.1.1 Puls-meddelande

Puls-meddelandet skickas en gång per sekund och har TPDO 0x700 + nod-id. Det skickar en byte data som talar om vilket stadie BMS befinner sig i enligt tabellen nedan.

Värde	Stadie
0x00	Initiation
0x7F	Pre-operational
0x05	Operational

När BMS är i initiation har den nyss startat och håller på att initiera allt som behövs för att alla funktioner i BMS ska fungera.

Pre-operational syftar på att allt är initierat men den har upptäckt att det finns fler enheter med samma nod-id på CAN-bussen och kommer sannolikt att byta nod-id. BMS ändrar till operational om inte nod-id behövts ändrats på två sekunder.

När BMS har kommit till operational ska allt vara initierat och BMS ska ha ett unikt nod-id på CAN-bussen.

7.1.2 Tilldelning utav nod-id med Serienummer

Varje nod som är kopplad till CAN-bussen kommer skicka ut NMT meddelanden kontinuerligt. CAN-bussen är initierad med TPDOer vilket gör att alla enheter kommer se meddelanden men endast då en nod upptäcker att meddelandet innehåller samma nod-id som sitt eget kommer detta trigga en funktion som ändrar nod-id, se avsnitt 3.7.4.

Denna funktion ändrar nod-id baserat på batteriets serienummer. Serienummrets första siffra läses in från eeprom. Därefter adderas siffran i serienummret till

nod-id och sparas till eeprom på den adress där enhetens nod-id är sparat. Som tidigare nämnt görs detta för att inte behöva upprepa processen varje gång systemet startas. Mottagarfilter uppdateras sedan och nod-id tilldelas värdet NodeID + (1,2,3..9). Man kan alltså på detta sätt ta fram nio olika nod-id. Dock skall bara nod-id mellan 42-49 användas till BMS vilket alltså innebär att sju möjliga nod-id finns. Därför måste nod-id alltid räknas om ifall det skulle bli över 49. De gånger nod-id blir större än 49 testas nod-id om från början med start på 42, då finns chansen fortfarande att ett nod-id tilldelas istället för att endast vänta på lägre siffror ur serienummret. Skulle det nya nod-id vara upptaget kommer hela funktionen att upprepas när nästa nod-id krock upptäcks. Dock med skillnaden att nästa siffra i serie numret då kommer att användas för tilldelning. Efter att hela serienummret gåtts igenom kommer funktionen starta om från början. Detta kan inträffa ifall fler än två enheter finns i samma system.

7.1.3 Tilldelning utav nod-id med A/D omvandling

När detta förslag upptäcker att det finns fler enheter med samma nod-id utförs en A/D omvandling på en analog ingång. Med resultatet från A/D omvandlingen skapas ett nytt nod-id genom att först kolla på första biten av resultatet, om biten är 0 så ändras inte nod-id och om biten är 1 så adderas nuvarande nod-id med 1. Om det fortfarande finns fler enheter med samma nod-id så används nästa bit på samma sätt. Detta fortsätter tills BMS har fått ett unikt nod-id eller hela resultatet ifrån A/D omvandlingen används. Om hela resultatet använts så gör en ny A/D omvandling och hela processen upprepas.

7.2 Kod översättning

7.2.1 BMS programvara

Programvara har översatts till C och funktioner har mestadels fått behålla samma struktur som tidigare. Programvaran finns nu i C som är ett vanligt förekommande programspråk generellt och har använts länge och inom många olika områden [12]. Fördelar med mjukvaran sett till systemets prestanda är dock inte testade. Beroende på den långsamma exekveringstiden för BASIC-program gemfört med c-program kan man med säkerhet anta att programmet blir snabbare.

7.2.2 LCD display med CAN mottagare

För test utav systemet användes en LCD display som visade vad som skickades på CAN bussen. Programvaran för denna fanns först bara i BASIC men blivit översatt till C. Som en biprodukt utav testerna har denna fått utökad funktion. Tidigare fanns bara funktioner för att skriva tal med olika många stora siffror och en bakgrunds funktion. I projektet skapades funktioner för att skriva ord, tal med små siffror och en byte binärt.

8 Slutsats

Detta projekt har resulterat i en mjukvara i C för en BMS baserat på tidigare mjukvara skriven i BASIC. BMS har också utökats med funktionalitet som innebär att upp till nio olika batterinoder kan användas på samma CAN buss. Företaget kan använda mjukvaran som den är eller använda den som grund att arbeta vidare på och förbättra.

Eftersom arbetet skulle utföras på en befintlig krets innehöll arbetet att analysera och förstå BMS. Mycket arbetstid lades även på att förstå kommunikationsprotokollen som ingick i systemet. Då det inte var så enkelt som att endast översätta programspråk för dessa. Det tog därför mer tid än väntat att översätta kod för styrning av skärmar.

För att ta fram en lösning till batteriernas nod-id behövdes CAN och CANopen studeras. Om CAN nätverket kan de konstateras att det är ett brett område som kan utvecklas mycket för den implementerade lösningen. Dock är funktionerna för nod-id tilldelning fullt fungerande. Vid systemstart får batterinoderna olika nod-id som var tänkt.

Projektets försenade start innebar att viss utveckling av BMS ej påbörjades. Clean Motion var intresserat av ett MCU-byte som skulle medföra omritning av kretskort i CAD samt mjuvaruanpassning vilket inte har genomförts.

8.1 Diskussion

8.1.1 CAN nod-id tilldelning

Det finns för och nackdelar med båda förslagen för CAN nod-id tilldelning. I nuläget passar förslaget med A/D omvandling bäst för denna BMS då serie numrets format inte är helt bestämt. Nackdelen med detta är dock om det kan inträffa att två enheter kan ha samma konstanta värde på den analoga ingången kan inte enheterna få olika nod-id. Om det andra förslaget används vet man att serie numret är unikt och därför kommer ett unikt nod-id hittas.

Denna funktion borde testas med fler än två enheter för att säkerställa funktion för flera enheter.

När BMS är i pre-operational state, alltså när den bestämmer nytt nod-id, skulle den kunna skicka puls-meddelanden oftare än ett varje sekund för att det ska gå snabbare att bestömma nytt nod-id.

8.1.2 Utökade funktioner för display

De nya funktionerna för LCD displayen var inte något mål med projektet utan mer en biprodukt utav testerna. Dessa kan dock vara användbara för att skriva

nya testprogram i framtiden eller om denna display ska användas på annat sätt i framtiden.

8.1.3 Miljöpåverkan

Detta projekt har inte någon direkt miljöpåverkan då det inte skapar någon produkt som varken är bra eller dålig produkt. Dock kan projektet ha en indirekt positiv inverkan på miljön då det utvecklar en redan befintlig produkt, Zbee som är bättre för miljön än många andra alternativ till den. Detta projekt kan göra Zbee mer tillgänglig och attraktiv och på det sättet påverka miljön positivt.

8.1.4 Projektets genomförande

Projektet slutfördes med acceptabelt resultat trots projektets försenade start som bl.a beror på att det tog tid att komma överens med företaget och skolan om en uppgift och frågeställning med lämplig storlek. Dock efter att projektet startades så fungerade arbetet väl. BMS kan nu användas med - och fortsätta utvecklas med uppdaterat programspråk.

Referenser

[1] *I2C-bus specification and user manual*
NXP Semiconductors, 2014.
Tillgänglig: `https://www.nxp.com/docs/en/user-guide/UM10204.pdf`
[Hämtad: 2018-05-09]

[2] Wikipedia
Tillgänglig:`https://commons.wikimedia.org/wiki/File:OSI_Model_v1.svg`

[3] Keith Pazul, *Controller Area Network (CAN) Basics*,
Microchip Technology Inc, 2002.
Tillgänglig:`http://ww1.microchip.com/downloads/en/AppNotes/00713a.pdf`.
[Hämtad: 2018-04-13]

[4] Steve Corrigan, *Introduction to the Controller Area Network (CAN)*
Texas Instruments, 2002 – reviderad 2016.
Tillgänglig: `http://www.ti.com/lit/an/sloa101b/sloa101b.pdf`
[Hämtad: 2018-04-16].

[5] *Controller Area Network Physical Layer Requirements*
Texas Instruments
Tillgänglig:`http://www.ti.com/lit/an/slla270/slla270.pdf`

[6] National Instruments, *The Basics of CANopen.* aug 21, 2013.
Tillgänglig:`http://www.ni.com/white-paper/14162/en/`.
[Hämtad: 2018-04-09].

[7] CAN in Automation - Knowledge database.
Tillgänglig:`https://www.can-cia.org/can-knowledge/`
[Hämtad: 2018-04-16].

[8] *CAN Specification v2.*
Robert Bosch GmbH, 1991.
Tillgänglig:`http://esd.cs.ucr.edu/webres/can20.pdf`
[Hämtad: 2018-04-03].

[9] *The OSI Model's Seven Layers Defined and Functions Explained.*
Tillgänglig:`https://support.microsoft.com/sv-se/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained`
[Hämtad: 2018-04-18]

[10] Olaf Pfeiffer & Christian Keydel, *Challenges of CANopen Node ID assignment, avoiding duplicates.*
CAN in Automation, 2013.
Tillgänglig:`https://www.can-cia.org/fileadmin/resources/documents/proceedings/2013_mmc_pfeiffer.pdf`
[Hämtad: 2018-05-09]

[11] *SPI Block Guide v3.06.*
Motorola, Inc 2000 - reviderad 2003.
Tillgänglig:https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf
[Hämtad: 2018-05-11]

[12] *TIOBE Programming Community Index for MAY 2018*
Tillgänglig:https://www.tiobe.com/tiobe-index//
[Hämtad: 2018-05-20]