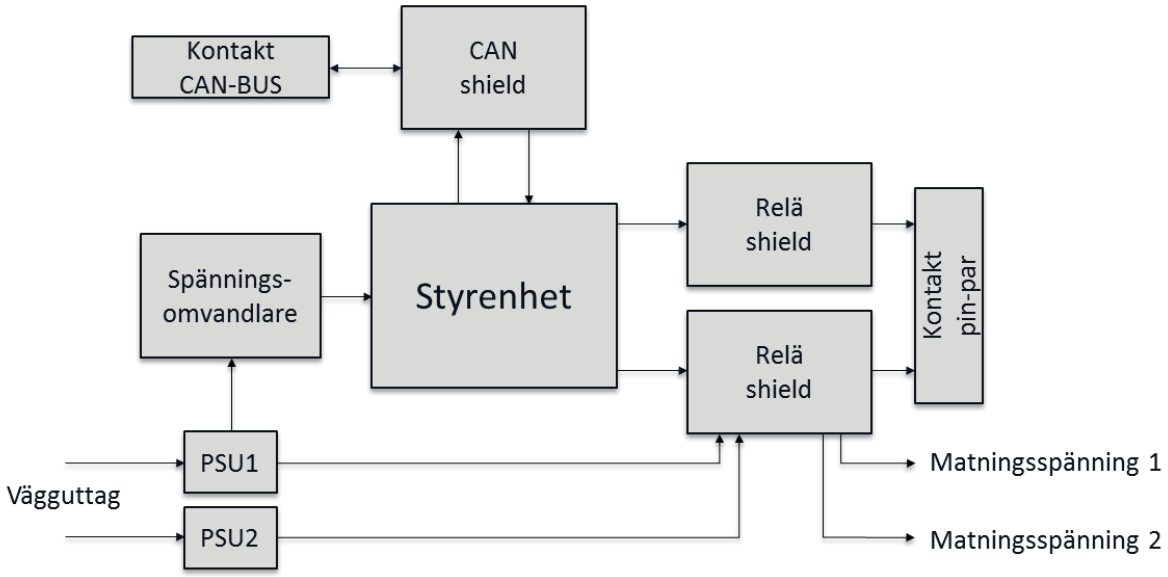
CHALMERS



Automatisk testning av elektriska styrenheter

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

CECILIA HERNQVIST
STINA JANGE

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

Automatisk testning av elektriska styrenheter
Examensarbete inom Högskoleingenjörsprogrammet i Mekatronik

CECILIA HERNQVIST
STINA JANGE

Institutionen för signaler och system
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telefon +46 (0)31-772 1000

Omslagsbild: Blockschema
Skapad av författarna 2015-05-25

Göteborg, Sverige 2015

Förord

Denna rapport presenterar vårt examensarbete som har utförts vid institutionen Signaler och system på Chalmers tekniska högskola, och är avslutande del i utbildningen Mekatronikingenjör (180 hp). Examensarbetet omfattar 15 hp och har under 10 veckor genomförts på Consat Engineering AB i Göteborg.

Vi vill rikta ett stort tack till våra handledare på Consat, Jonas Williamsson och Peter Brunnsåker, för det stora stödet och för tålamodet med alla våra frågor. Ett lika stort tack riktas till vår handledare på Chalmers, Morgan Osbeck, som har korrekturläst denna rapport under arbetets gång och gett lämplig kritik samt kommit med tips och råd.

Vi vill även tacka Måns Breitholtz, Andreas Johansson, Magnus Lindén, Jakob Mattsson samt Mats Nordström på Consat för den hjälp och handledning som erhölls under examensarbetet.

Sammanfattning

Testning av styrkretsar i fordon är idag till stor del automatiserat. Dock finns det fortfarande vissa hårdvarutester, såsom kortslutningar samt styrning av spänningstillförsel, som utförs med hjälp av manuella tilltag. Arbetet som beskrivs i denna rapport har under 10 veckors tid utförts på Consat Engineering AB i Göteborg, och syftet har varit att ta fram en hårdvarumodul som via en kommunikationsbuss skall kunna tolka in ett styrmeddelande för att sedan utföra aktuellt test på ett anslutet kretskort. Detta görs för att säkerställa att kretskortet som testas skriver ut rätt felkod då ett visst hårdvarufel genereras. I frågeställningen ingår att göra en förstudie kring lämplig huvudmodul för hårdvaran, samt kring övrig hårdvara som kan tänkas behövas. Därefter har modulen byggts upp och kopplats hårdvarumässigt samt programmerats för att kunna utföra de tester som ställdes upp som krav, där en av funktionerna innebär att modulen kan styra tillförseln av matningsspänning ut till kretskortet från två fristående spänningsaggregat där spänningen till kretskortet kan stängas av eller sättas på. Nästa moment innebär att modulen kan generera kortslutning på olika pin-par på kretskortet, där pin-paren har kopplats in till modulen. Den modul som har tagits fram är en prototyp och tanken som har genomsyrat projektet är att modulen skall ha stor utvecklingspotential och i framtiden utvecklas till en beständig produkt.

Summary

Tests to verify desired function of electronic control units in vehicles are often executed automatically today. However, there are still some hardware-tests, such as short circuits or regulation of voltage supply, that need some manual handling to be performed. The project described in this report has been carried out during 10 weeks at Consat Engineering AB in Gothenburg, and the main intention has been to develop a hardware-module that should be able to receive a control message by bus-communication. It should then process the message and perform given tests at the connected electronic control unit. These tests are done to ensure that the control unit prints the right error code when a hardware failure occurs. The question formulation involves fulfilling a pre-study about appropriate main-hardware, and other hardware that will be needed. Thereafter the module has been assembled and programmed to be able to perform the tests that were stated as a demand. In Step 1 the main object is to be able to control the supplied voltage to the control unit from two detached power supplies where the voltage to the control unit may be set to on or off. In Step 2 the module should be able to generate a short circuit at different pair of pins at the control unit, whereas the pairs have been connected to the module. The finished module is a prototype and through the whole project the thought has been that the module should have potential for further development, and in the future be developed as an implemented product.

Innehåll

Beteckningar

AC	Alternating Current
CAN	Controller Area Network
COM	Common
DC	Direct Current
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
GPIO	General-purpose input/output
HD	High-Definition
HDMI	High-Definition Multimedia Interface
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
LED	Light-Emitting Diode
MISO	Master Input, Slave Output
MOSI	Master Output, Slave Input
NC	Normally Closed
NO	Normally Open
OBD	On-Board-Diagnostics
PWM	Pulse-Width Modulation
PSU	Power Supply Unit
SCK	Serial Clock
SD	Secure Digital
SPI	Serial Peripheral Interface
SS	Slave Select
USB	Universal Serial Bus

Nedanstående namn förkortas hädanefter enligt

Consat	Consat Engineering AB
Teknisk handledare	Teknisk handledare Peter Brunnsåker på Consat Engineering AB
Huvudhandledare	Huvudhandledare Jonas Williamsson på Consat Engineering AB
Chalmers handledare	Chalmers handledare Morgan Osbeck Institutionen för signaler och system

1 Inledning

Detta arbete handlar om att sätta upp en hårdvarumodul och programmera denna till att utföra automatiska hårdvarutester av styrenheter i fordon. Nedan förklaras grunderna i arbetet och varför det har utförts.

1.1 Bakgrund

Consat Engineering AB är ett ingenjörs- och teknikkonsultbolag som har varit verksamt sedan 1986. Consat Engineering AB har kontor i Stockholm, Göteborg och Montreal i Kanada och driver uppdrag inom områdena mekanikkonstruktion, elektronik och mjukvara, telematik samt miljöteknik.

Som leverantör av bl.a. mjukvara till olika styrenheter inom fordonsindustrin krävs av Consat Engineering AB att utföra tester av funktion och kvalitét. För att detta skall kunna göras så effektivt som möjligt krävs automatisering av tester som i nuläget är manuella.

1.2 Syfte

Syftet med examensarbetet är att programmera hårdvara för att kunna automattesta elektriska styrenheter i bilar då typiska hårdvarutest såsom kortslutning och variation i inspänning fortfarande utförs manuellt. Efter avslutat examensarbete är tanken att Consat Engineering AB skall ha en fungerande enhet för generella tester som sedan skall kunna utvecklas vidare samt omformas för att passa specifika krav för exempelvis olika bilmodeller.

1.3 Avgränsningar

Examensarbetet är på 15 högskolepoäng och utförs under tio veckor. Då projektet är tidsbegränsat kommer nödvändig hårdvara köpas in som färdiga produkter. Modulen som skapas kommer vara en prototyp, att bygga in den i en sluten låda och presentera en färdig produkt är ej prioriterat utan kommer utföras i mån av tid. Under projektet kommer det heller inte tas hänsyn till ekonomiska aspekter.

1.4 Precisering av frågeställning

Hårdvarumodulen skall vid signal via kommunikationsbuss generera aktuellt hårdvarutest på ett kretskort som kopplas in till modulen, där testet består av att få bekräftelse på att rätt felmeddelande genereras vid hårdvarufel på det anslutna kretskortet. Projektet skall behandla följande punkter vid genomförandet.

• Förstudie med specifikation och tidplan

- Systemdesign
- Tidplan
- Hårdvara
- Programspråk
- Kommunikationsgränssnitt
- Funktionalitet
    - Steg 1: Spänning on/off
    - Steg 2: Generera kortslutning
    - Steg 3 (om tid finnes): Variera spänning

• Utvecklingsarbete

- Uppsättning av hårdvara
- Programmering av hårdvara
- Teknisk rapport

Beroende på hur projektet ligger till tidsmässigt kan fler funktioner komma att läggas till efterhand, utifrån företagets önskemål.

2 Teknisk bakgrund

För att läsaren av denna rapport skall erhålla tillräckliga kunskaper för att kunna tillgodogöra sig rapportens innehåll följer i detta kapitel information som ligger till grund för projektet, samt information angående de komponenter och tekniska termer som används.

2.1 Fordonsdiagnostik och dess syfte

Fordonsdiagnostik används för att kunna övervaka och kontrollera ett fordons inre system, där diagnostiken ofta består av att läsa av sensorer och få information ifrån de styrenheter som det elektriska systemet består av. Ett exempel är att det i många moderna bilar finns noder som mäter miljöpåverkan av bilen, såsom kontroll att utsläppen inte överstiger de nivåer som finns satta enligt lagen. Fordonsdiagnostik spelar även en viktig roll säkerhetsmässigt då informationen som fås ifrån styrenheterna kan indikera i ett tidigt stadie ifall någon av bilens funktioner inte agerar som de ska. I forskningssyfte används fordonsdiagnostik för att samla in mätdata som sedan kan användas vid utveckling av nya bilar, till exempel bilar som har mindre miljöpåverkan eller som bättre klarar en farlig situation vilket ökar trafiksäkerheten samt säkerheten för passagerarna i fordonet. Det går också att mäta hur olika komponenter slits i fordonet så att en varning ges i god tid då komponenten behöver kontrolleras eller bytas ut. Därför är det också viktigt att kontrollera att olika varningssystem och felmeddelanden i bilen fungerar korrekt och skrivs ut på rätt sätt, för att undvika större brister och se till att bilen framförs på ett säkert sätt. (Bylund, 2009)

2.2 Testanläggning

För att få en tydligare bild av hur Consat utför tester i dagens läge hölls en intervju med den tekniska handledaren. Nedan följer en beskrivning av hur testerna går till på företaget samt övrig nödvändig information.

Consat erhåller en specifikation med en lista på krav över hur ett kretskort skall fungera, där listan kan innefatta närmare 2000 olika krav som behöver verifieras. Kraven är indelade i olika kategorier där DTC, Diagnostic Trouble Code, är en av dem. De automatiserade testen av hårdvara som under detta projekt skall utvecklas faller inom denna kategori. Utifrån programmet CANoe, som är ett verktyg till för att utveckla, testa och diagnostisera styrenheten för ett fordon (ECU), testas de krav som finns med i specifikationen. Detta program kan simulera alla signaler i bilen (exempelvis sätesvärmare, bältesvarnare, om en bildörr är öppen eller stängd etc.) och används för att upptäcka eventuella problem i ett tidigt stadie och åtgärda dem.

Kommunikationen sker via fältbusskommunikation och informationsdata sänds mellan bilens olika noder. Consat testar en individuell nod, som sedan kommunicerar med bilens alla övriga noder. CANoe stödjer olika typer av fältbussystem, däribland CAN och FlexRay, vilka är de som används i det här fallet. FlexRay är det huvudsakliga fältbussystemet som sköter kommunikationen mellan huvudnoderna, och CAN sköter kommunikationen på vardera underliggande nät av noder som styrs av respektive huvudnod. En huvudnod styr olika områden i bilen och de underliggande noderna sköter en specifik uppgift inom det området.

Consats uppkopplingsnod, som är en huvudnod, styr övergripande kommunikation i bilens elsystem. Uppkopplingsnoden skickar ut meddelande till andra noder via höghastighetsbussen FlexRay och Consat testar på så vis kretskortet för att se om det kommer fram ett meddelande eller ej. Fordonsdiagnostiken sker via denna uppkopplingsnod och då en bilmekaniker sedan kopplar in sin dator till bilens system läses eventuella felkoder in som jämförs mot en manual.

Detta möjliggör enkel detektering av fel då det i manualen finns listat var ett specifikt fel har uppkommit vid given felkod.

Då Consat testar ett krav utifrån erhållen specifikation vet vederbörande på förhand vilken felkod som skall skrivas ut. Genom att sedan simulera ett fel går det att kontrollera att rätt felkod faktiskt skrivs ut av systemet. Varje test har ett utförande som skall göras *action* och ett förväntat resultat *expected* vilket resulterar i antingen PASSED eller FAILED.

Vissa tester av krav kräver endast en uppstart medan andra tester kräver att manuella moment utförs samtidigt som programmet för just detta test körs. Detta kan ske genom att programmet säger till att en viss pinne skall kortslutas och väntar tills detta utförts uppföljt av ännu en begäran.

Varje testfall ger ut en statusrapport och då det uppstår buggar som kräver någon sorts mjukvaruförändring läggs koderna in i Consats ärendehanteringsprogram Redmine. Då ser andra utvecklare inom projektet att det finns en bugg som bör åtgärdas. Testfallen visar vilket steg som gått fel och markerar detta, och därefter samlas varje testomgång i en rapport för det aktuella mjukvarusläppet. (Brunnsåker, 2015)

2.3 Datakommunikation

Nedan följer förklaringar kring de kommunikationsbussar som används i projektet.

2.3.1 CAN

CAN, som är en förkortning av *Controller Area Network*, introducerades under början på 80-talet av Robert Bosch GmbH. CAN-systemet är en fältbusskommunikation som i första hand utvecklades för bilindustrin, men som under senare tid även har kommit till användning inom andra områden såsom tåg, flygelektronik, militären m.m. (Lawrence, 2013)

CAN möjliggör meddelandesändning mellan flera noder (datakommunikation) eller styrenheter i fordonet på ett snabbt och säkert sätt. Ett meddelande innehåller identifierare, data samt felkontrollsdata. Då en mottagare läser identifieraren avgör denna om meddelandet skall tas emot eller ej. Meddelandets identifierare avgörs av 11 bitar (där 11 bitar är standardformat och 29 bitar utökat format) och data av 0-64 bitar. Bitarna i ett meddelande är antingen dominanta (nollor) eller recessiva (ettor). Det finns fyra typer av meddelanden i CAN; *data frame*, *remote frame*, *error frame* och *overload frame*. Den förstnämnda meddelandetypen innehåller datainformation som skickas till en eller flera mottagare. Meddelandesändningen går ut till alla noder och det är innehållet av identifierare som avgör vem eller vilka som fångar upp det. Då meddelandetypen *remote frame* skickas av en nod begärs en sändning av viss data från sändaren. Då det uppträder ett fel i CAN-protokollet upptäcks dessa av en nod som sedan skickar meddelandetypen *error frame*. Detta meddelande uppfattas av alla noder då det består av sex dominanta bitar i rad, vilket annars inte förekommer i en *data frame* eller *remote frame*. Den sistnämnda meddelandetypen *overload frame* används för att mottagarna skall få en tidsfördröjning innan meddelandetyperna *data frame* eller *remote frame* skickas på nytt. (Osbeck, 2014)

2.3.2 SPI

SPI-bussen är en synkron seriekommunikation som används av mikrocontrollers för att kommunicera med andra enheter eller en annan mikrocontroller. Det finns en masterenhet, vanligen en mikrocontroller, och en slavenhet då SPI-protokollet används. Bussen tillämpar fyra logiska signaler; MISO (data från slavenhet till masterenhet), MOSI (data från masterenhet till slavenhet), SCK (klocksignaler som genereras av masterenheten) samt SS (slave select). SS har olika pin-utgångar beroende på vilken enhet som skall kommunicera med mastern. SS måste sättas till utgång och då pinnen sätts till HIGH kan kommunikation ske med masterenheten, då pinnen sätts till LOW ignoreras eventuella signaler från masterenheten. (Arduino, 2015)

2.4 Hårdvara

För att få förståelse kring hårdvaran som används i projektet följer nedan beskrivningar kring flertalet komponenter som på något sätt har tillämpats under tidens gång.

2.4.1 Reläer

Ett relä är en elektriskt manövrerad strömställare där en spänningsnivå kan användas för att bryta/sluta en krets vid annan spänningsnivå. Ett relä används för att sluta för att under den aktiva tiden föra över ström eller spänning i en koppling. (Gurevich, 2006) Ett exempel på ett reläs användningsområde är i en strömbrytare där reläet aktiveras vid tillslag och avaktiveras vid frånslag.

2.4.2 Relä shield

Till både mikrocontrollers och mikrodatorer finns det olika relämoduler (shields) som kan kopplas samman med huvudkortet för att bygga till en viss funktionalitet. En relä shield är ett sådant kort som innehåller ett eller flera reläer som kan manövreras från en styrkrets till att styra önskade funktioner. Förutom antal reläer så finns det olika varianter av relä shields där t.ex. matningsspänningen kan variera. (Spark docs, 2015)

2.4.3 CAN-shield

Då en CAN-shield kopplas ihop med en styrmodul sker kommunikation mellan de två plattformarna. CAN-bussmodulen använder sig av SPI-protokoll för att kunna skicka data mellan mikrocontrollern och CAN-shielden. Genom att använda sig av en CAN-shield går det sedan att kommunicera via CAN med andra utomstående komponenter. (Anderson & Cervo, 2013)

2.4.4 Kopplingsbräda

En kopplingsbräda, eller ett kopplingsdäck, används vid prototypkopplingar eller icke-permanenta kopplingar, då man på kopplingsbrädan kan koppla komponenter utan att behöva löda fast någonting. Kopplingsplattan innehåller rader där samma rad är sammankopplad genom metallremsor på baksidan. Det medför att alla komponenter eller sladdar som är kopplade till samma rad är kopplade till varandra, och dessa kan i sin tur ansluta till kompontenter på en annan rad genom att tråda emellan. Många kopplingsbrädor har även så kallade diken mellan raderna som särskiljer dessa så att det går att sätta en komponents olika ben över dikena utan att benen är anslutna till varandra. (Sparkfun, 2015)

2.4.5 Spänningsregulator

En spänningsregulator eller spänningsomvandlare som det även kallas kan användas då man vill modifiera spänningen i en del av en krets. Ett exempel är om en spänningskälla med högre spänning skall användas för att driva en krets där en eller flera komponenter bara klarar av en lägre spänning. Då krävs en spänningsregulator som omvandlar spänningen till den lägre nivån. En sådan regulator som omvandlar från hög till låg kallas *step down*, medan en regulator som fungerar tvärtom kallas *step up.* (Britannica Academic, 2015) Den spänningsregulator som används i detta projekt är en DC/DC-omvandlare och klarar inspänning på mellan 9 V och 36 V och skickar ut utspänning på 5 V. (Digikey, 2015)

2.4.6 Hylslist

För att underlätta kretskortsmontage finns stackningsbara hylslister. Hylslister finns med olika antal poler och benavstånd och är ofta förekommande inom elektronik. De kan lödas fast på kretskort och ansluts sedan via polerna till ett annat kort. (Electro:Kit, 2015)

2.4.7 D-subkontakt

D-subkontakt (även kallad D-subminiatyrkontakt) är en typ av elektrisk kontakt som ofta förekommer i datorsammanhang. Det finns olika modeller utrustade med olika antal poler. En D-subkontakt utrustad med pinnar kallas hankontakt (engelska *plug*) och en D-subkontakt med hylsor kallas honkontakt (engelska *socket*). (Wikipedia, 2015)

2.4.8 Operationsförstärkare

En operationsförstärkare, eller en OP-förstärkare som det ofta kallas, är en linjär elektrisk komponent som historiskt sett ofta har använts för att utföra logiska matematiska operationer. Nu för tiden har dock OP-förstärkare ett bredare användningsområde och kan beroende på uppbyggnad både höja och sänka spänningen eller strömmen i en krets. Vanligtvis består OP-förstärkaren av en minusingång, en plusingång och en utgång, samt två pinnar för positiv och negativ matningsspänning, där den negativa matningsspänningen kan ha ett negativt värde eller vara satt till 0 V. (Electronics Tutorials, 2015)

2.4.9 Digital potentiometer

En potentiometer är en typ av reglerbart motstånd med tre anslutningar, där det mellan två av anslutningarna sitter ett motståndelement och den tredje är kopplad till en släpkontakt eller glidkontakt som kan flyttas längs med motståndsbanan. En potentiometer fungerar som en spänningsdelare och en digital potentiometer fungerar på samma sätt som en analog potentiometer, men styrs digitalt från en mikrocontroller (eller annan dylik enhet) istället för manuellt. Digitala potentiometrar styrs vanligtvis via seriebusskommunikation såsom SPI, GPIO eller I^2C.

En digital potentiometer kan tillämpas inom flera användningsområden då en variabel eller parameter behöver regleras eller kontrolleras, exempelvis reglage av spänning, ström, resistans, induktans, kapacitans, frekvens osv. Den kan även användas som en digital-till-analog-omvandlare. (Onsemi, 2013)

2.4.10 Programmerbart nätaggregat

Ett programmerbart nätaggregat kan generera driftspänning eller driftsström till elektroniska anordningar beroende på vad de ingående komponenterna behöver. Det programmerbara nätaggregatet använder vanligtvis en integrerad mikrodator för att styra och kontrollera strömkällan. Då nätaggregatet nås av en programmeringssignal tillhandahålls önskad uteffekt. (Lanni, 2008)

2.5 Mjukvara

Nedan ges kort beskrivning kring programvara och övrig mjukvara som har koppling till projektet.

2.5.1 Högnivåspråk

Med högnivåspråk menas inom programmering kodning som är mindre abstrakt än datorns eget maskinnära sätt att hantera information, vilket i grund och botten hanteras av ettor och nollor. Högnivåspråk ämnar till att mer likna en människas hantering av problemet som skall utföras, vilket underlättar lösning på problem av mer komplex natur då användaren med högnivåspråk ej behöver ta hänsyn till exakta minnesadresser och register. (Teufel, 1991)

2.5.2 C-programmering

C-kod är ett programmeringsspråk som kombinerar maskinnära instruktioner med struktur från högnivåspråk. Det har under åren skapats olika standardiseringar kring C-kod varav C89 och C99 är de som används kommersiellt. Trots att C99 är en nyare standardisering så är C89 fortfarande väldigt utbrett då många kompilatorer fortfarande inte är kompatibla med C99. Ur C har flera högnivåspråk senare skapats, bland annat C++ och Objective C. (Bilting & Skansholm, 2011)

2.5.3 Busmaster

Busmaster är ett verktygsprogram som kan användas för att simulera databusskommunikation, exempelvis CAN-kommunikation, där programmet agerar som en nod uppkopplad till ett nätverk. Programmet kan användas för att simulera och analysera meddelanden som skickas via bussen. Programmet är open source vilket öppnar för stora utvecklingsmöjligheter kring funktionaliteten, men standard är att använda programmet i det utförande som finns nedladdningsbart. (Busmaster, 2015)

2.6 Pulse-Width Modulation

Pulse-Width Modulation, förkortat PWM, står för pulsbreddsmodulering. En PWM-signal ser ut som en fyrkants-signal och genererar spänning genom att snabbt skifta mellan hög och låg, vilket sker i pulser. Den höga flanken i fyrkantsvågen är på och den låga flanken innebär av. Genom att använda PWM går det att välja hur hög spänning som skall skickas ut genom att medelvärdet som pulserna skapar är det som matas ut. Detta påverkas av pulsernas frekvens samt bredden på pulserna. (Microchip Technology Inc, © 2010-2012)

3 Metod

Som uppstart till projektet skrevs en inledande planeringsrapport som innehöll *bakgrund, syfte, avgränsningar* samt *precisering av frågeställning* för projektet. Efter godkännande av planeringsrapporten från Chalmers handledare samt ansvariga på Consat sattes en tidsplan upp i form av ett Gantt-schema för att få en överblick över tioveckorsperioden och möjliggöra en första planering kring vilka moment som behövde utföras samt under vilken tidsperiod de beräknas utföras. Schemat användes även senare för att bryta ner arbetsuppgifter i mindre delmoment vilket underlättar metodiskt och organiserat arbete. En kurs i informationssökning gavs av Chalmers i början av projektet där genomgång hölls om hur man på bästa sätt söker fram information samt hur källhänvisning skall gå till. Ganska tidigt under perioden hölls också genomgång av Consat kring hur de arbetar med ärendehantering i programmet Redmine, samt versionshantering av mjukvara i programmen Git och Gerrit. Sådan metodisk arbetsgång tillämpades i projektet för att få det att vara så likt ett riktigt arbetsprojekt som möjligt.

Arbetet start bestod av informationssökning för att bestämma hur utrustningen skulle byggas upp. Där ingick att både bestämma hårdvara samt mjukvara, där val av mjukvara till stor del påverkades av valet av hårdvara. I och med detta togs en kravspecifikation fram där krav och önskemål för projektet fanns med. För att på ett metodiskt sätt bestämma vilken sorts huvudmodul som skulle användas togs tre potentiella moduler fram i samråd med handledare på Consat, där dessa sedan jämfördes med varandra i en Pughmatris, eller relativ beslutsmatris som det också kallas. Utifrån huvudkomponenten valdes sedan kringliggande hårdvara samt vad för mjukvara som skulle användas. För att underlätta det fortsatta arbetet ritades utkast upp på tänkt flödesschema för de olika stegen i programmeringen. I och med de första flödesschemana togs också en målbild fram mer i detalj kring hur programmet bör vara uppbyggt och vad som faktiskt skall ske. För att även få en klar bild kring hur dessa tester utförs i dagsläget hölls en intervju i uppstarten av projektet med teknisk handledare på Consat (se avsnitt 2.2), där vederbörande fick förklara vad för tester som utförs och hur de genomförs.

Därefter inleddes utförandefasen, som bestod av att koppla upp hårdvara samt skriva programkod. När dessa testkopplingar upplevdes som klara fortgick utförandefasen där varje funktionalitet byggdes på och testades allt eftersom. Vidare lades programsekvenserna samman för att se till att funktionerna fungerade som en helhet. Vidare inleddes arbetet med CAN-kommunikationen där målet var att kunna ge kommando via CAN-bussen om de tester som skall utföras. Ett befintligt CAN-bibliotek som fanns att hämta hem från internet användes och den redan skrivna programkoden implementerades sedan tillsammans med CAN-biblioteket. Tester utfördes därefter även med CAN-funktionaliteten för att säkerställa att meddelanden kunde skickas på bussen och att funktionaliteten fortfarande var korrekt då kommandot efter förändringen skickades via CAN-bussen till styrkretsen. Verifiering av önskad funktion gjordes genom att använda programmet Busmaster, där simulering av signaler gjordes för att se att det både gick att skicka samt ta emot meddelanden till och från styrenheten.

För att i slutskedet göra en mer hållbar uppsättning av kretsen beställdes en aluminiumplatta, som samtliga komponenter monterades på. När uppsättningen var klar ritades ett kretsschema upp i KiCad över hur kretsarna är sammankopplade med varandra.

Varje vecka under tioveckorsperioden har en uppdatering kring pågående arbete skickats till handledare på Chalmers för att få lämplig kritik kring arbetet. Kontinuerligt har även utkast på rapporten skickats. Ett halvtidsmöte hölls även med handledare på Consat för att stämma av hur långt arbetet hade fortgått samt vad som var kvar att utföra. Under hela projektets gång har dokumentation utförts regelbundet både kring informationssökning samt utförandefas för att underlätta sammanställning av slutrapport.

4 Definition av målbild

Vid *precisering av frågeställning* nämndes tre funktionaliteter varav två utförandesteg som var obligatoriska och ett steg som var önskemål och som skulle utföras i mån av tid. Genom att ställa upp en kravspecifikation (se Figur 4.1) för projektet erhölls en överblick över vilka funktioner som är krav (K) och vilka som är önskvärda (Ö). För ytterligare förtydligan behövdes målet definieras vidare; vad är tänkt att utföras och hur skall det utföras. Målbilden har tagits fram i samråd med huvudhandledare och teknisk handledare på Consat. Då projektet är tidsbegränsat är det huvudsakliga målet att vid projektets slut ha en uppkopplad prototyp som fungerar enligt kravspecifikation. Om tid finnes kommer steg 3 att genomföras och kopplingen kommer sättas upp på en beständigare platta alternativt byggas in i en låda, som dock fortfarande skall vara möjlig att modifiera om ytterligare funktionalitet skall adderas.

För att stegen skall gå att utföra krävs CAN-kommunikation, där meddelande skickas via CAN om vilket hårdvarutest det är som skall utföras.

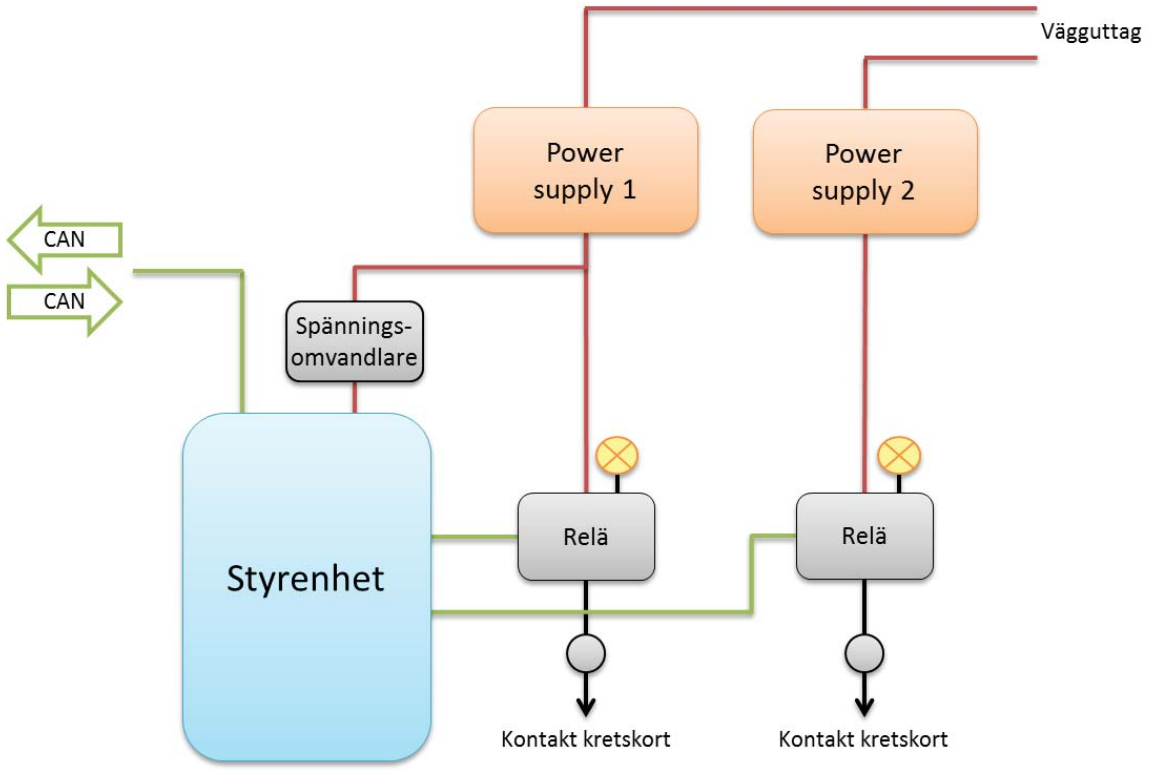
Chalmers	**Dokumenttyp** ***Projekt***	**Kravspecifikation** Automatisk testning av elektriska styrenheter
Utfärdare:	Cecilia Hernqvist Stina Jange	***Skapad:*** 2015-04-01 ***Modifierad:*** 2015-04-22

Funktion	***K/Ö***
Uppsättning lämplig hårdvara	K
Testa spänning on/off	K
Generera kortslutning	K
Variera spänning	Ö
Förfina uppsättning av hårdvara	Ö
Bygga in hårdvarumodul	Ö

Figur 4.1 - Kravspecifikation

4.1 Steg 1 – Spänning on/off

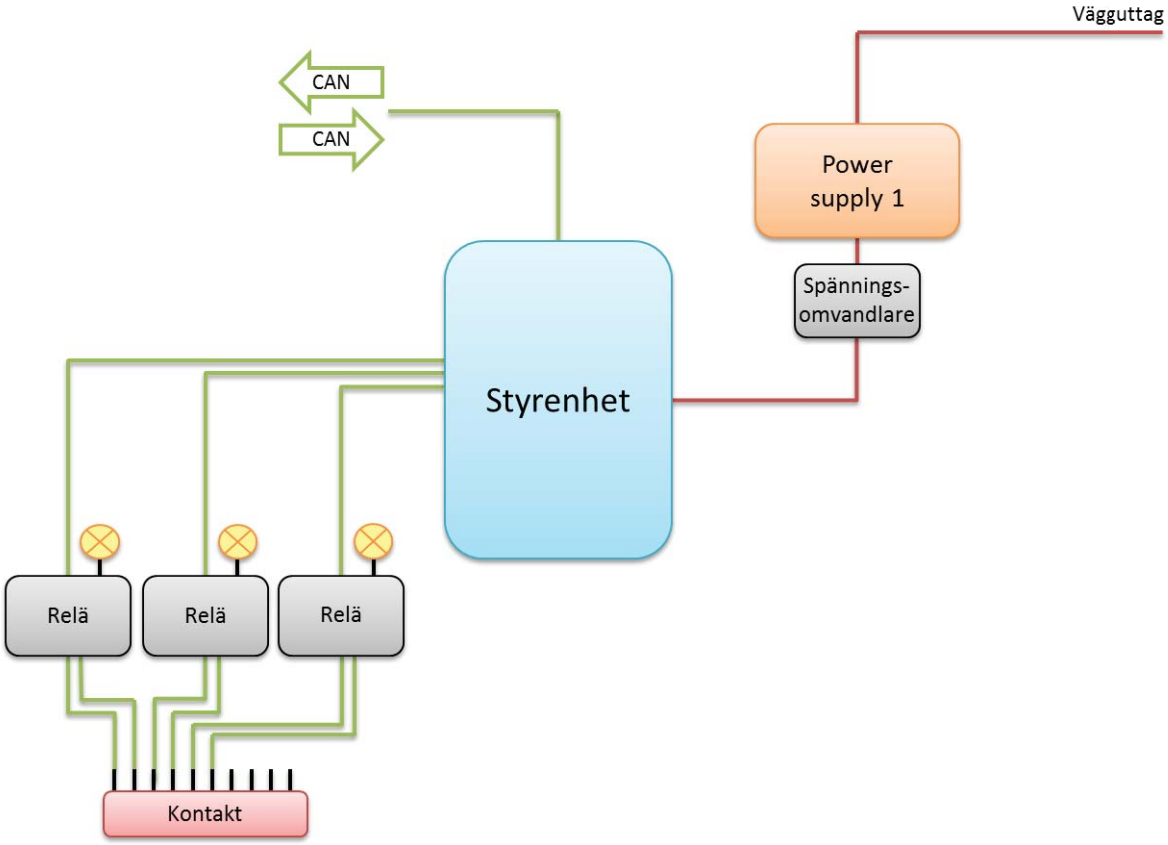
Första utförandefasen illustreras i figur 4.2 och handlar om att kunna styra tillförseln av spänning från ett eller två nätaggregat ut till det externa kretskortet som skall testas, där nätaggregaten kan symbolisera ett bilbatteri eller ett lastbilsbatteri. Det som skall utföras är att styrenheten vid CAN-kommando skall kunna stänga av eller sätta på spänningen från nätaggregaten ut till kretskortet. Detta sköts genom två reläer som vardera får styrsignal från styrenheten. Nätaggregaten är därmed inkopplade konstant med spänning som ”hänger” då reläet inte är aktiverat. Denna spänning kopplas sedan till kretskortet då reläet aktiveras. Tanken är även att styrenheten skall kunna drivas av ett av nätaggregaten via en spänningsomvandlare. De nätaggregat som i huvudsak kommer användas ligger på 13.8 V alternativt 24 V, där 13.8 V symboliserar ett personbilsbatteri och 24 V symboliserar ett lastbilsbatteri. På varje relämodul sitter åtta stycken lysdioder som ger visuell återkoppling om när spänningen är påkopplad eller ej.



Figur 4.2 - Överblick steg 1

4.2 Steg 2 – Generera kortslutning

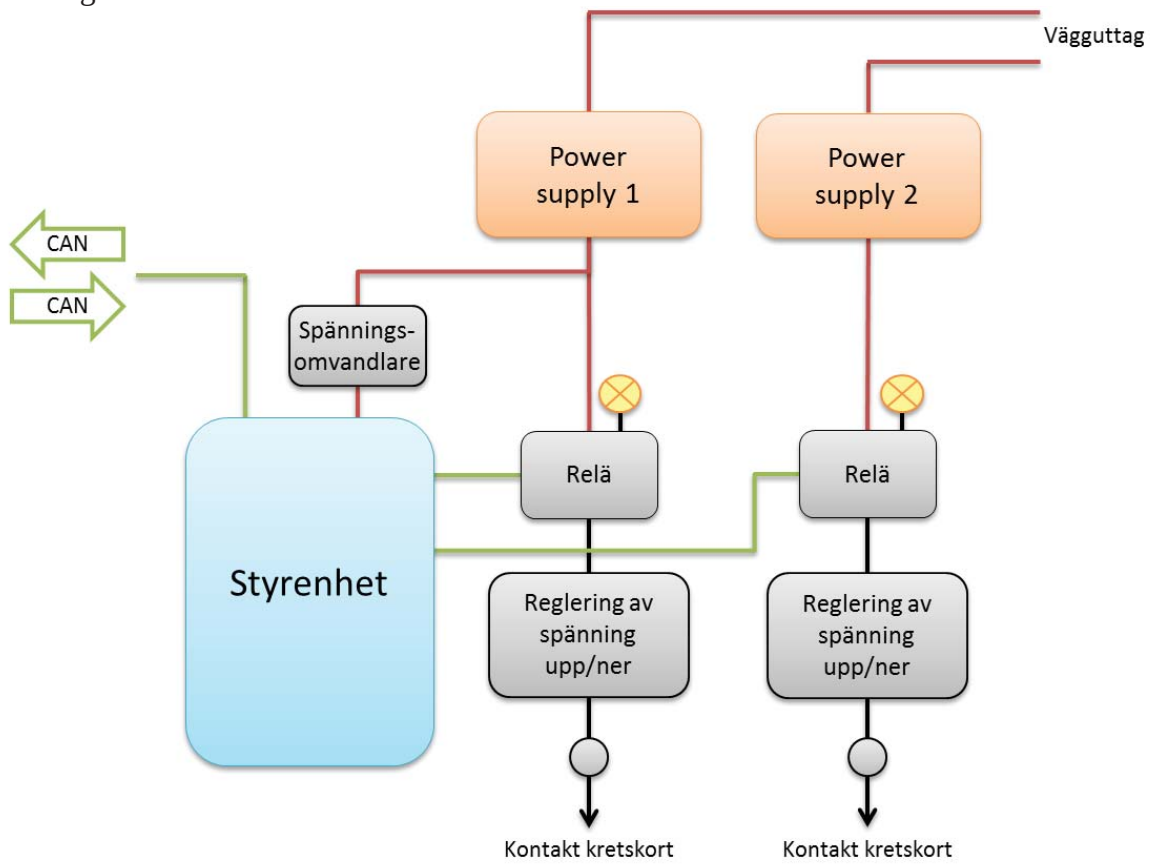
Nästa steg handlar om att kunna generera kortslutning mellan olika pin-par på kretskortet som testas. Detta realiseras genom att ett kontaktdon implementeras på modulen, där kontakten som kopplas in är anslutet till kretskortet som skall testas (se Figur 4.3). Via CAN-kommunikation skickas sedan meddelande om vilket pin-par det är som skall kortslutas. Detta sköts då av styrenheten som läser av meddelandet och ger styrsignal till reläet som hör ihop med det aktuella pin-paret, som aktiveras och därmed kopplar ihop de båda pinnarna, vilket genererar kortslutning. Även i detta steg ger lysdiod på reläet en visuell återkoppling om när spänningen är påkopplad eller inte. Detta steg kommer hållas relativt generellt för att göra det anpassningsbart till andra sorters kretskort än de som testas i dagens läge. Därmed kommer utrymme lämnas för inkopplande av fler reläer och möjlighet att använda fler pin-par än vad som används i detta projekt.



Figur 4.3 - Överblick steg 2

4.3 Steg 3 – Variera spänning

I tredje steget (se Figur 4.4), som är listat som önskemål, skall spänningen ut till kretskortet kunna varieras. Detta steg påminner om genomförandet av steg 1, dock med skillnaden att det krävs lämplig hårdvara för spänningsreglering mellan nätaggregaten och kretskortet som skall testas. Spänningsvariationen skall kunna utföras genom att meddelande skickas via CAN in till styrkretsen med information om hur spänningen skall regleras. Styrkretsen påverkar därefter den hårdvara som kontrollerar spänningen ut och ställer in spänningen till en viss nivå. Spänningen skall helst gå att reglera med steg som trappas upp eller ned 0.1 V per steg och det är fördelaktigt att ha en display kopplad till spänningen ut som visar vilken spänning som är inställd. Spänningen skall gå att variera mellan 0 V och 20 V, även om det hade varit önskvärt med ett maxvärde på 32 V för att kunna simulera batteri för personbilar men även för tunga fordon.



Figur 4.4 - Överblick steg 3

5 Val av utrustning

Första steget i projektet var att bestämma vilken hårdvara som skulle användas. I enlighet med projektets avgränsning skulle hårdvaran som används att bestå av befintliga produkter som kopplas samman för ändamålet. Det var därmed viktigt att inhämta information över de alternativ och möjligheter som finns på marknaden för att i slutändan kunna urskilja de produkter som passar slutmålet bäst. Första steget i detta var att bestämma vilken huvudkomponent som skulle användas, där den tekniska handledaren på Consat hjälpte till med att ta fram tre möjliga huvudkomponenter att välja utifrån. Dessa var Arduino, BeagleBone och Raspberry Pi. Därefter genomfördes litteratursökningar för att inhämta kunskap kring de olika alternativen samt för att få information om skillnader och likheter. De egenskaper som värderades högt var möjlighet till programmering i C, lämplig programvara, möjlighet till styrning av annan hårdvara samt antalet in-/utgångar alternativt möjlighet att implementera fler in-/utgångar.

5.1 Raspberry Pi

När föregångaren till det som idag är Raspberry Pi skapades var tanken initialt från medgrundaren Eben Upton att den skulle användas för att få upp barn och ungdomars intresse kring grundläggande programmering och digitalt skapande. Detta utvecklades och Raspberry Pi är numera en relativt välkänd mikrodator som bland annat kan användas som styrsystem vilket programmeras i högnivåspråk. (Upton & Halfacree, 2013) I linje med grundtanken så används Raspberry Pi mycket till att experimenteras med och skapa nya tekniska lösningar, både för privatpersoner samt för företag. En av anledningarna är det överkomliga priset samt de stora möjligheterna att ändra och lägga till funktioner. (Robinson & Cook, 2014)

5.1.1 Hårdvara

Raspberry Pi har släppts i olika versioner, där varje version sedan har genomgått en del revisioner där företaget har uppdaterat själva hårdvaran. En av de största skillnaderna mellan olika versioner är att på ett av de senare korten implementerades Ethernetkommunikation för att ge möjlighet till att koppla upp Raspberry Pi via internet. (Raspberry Pi, 2015) Mikrodatorn använder en processor från Broadcom, BCM2835 eller BCM2836, som är en processor som bland annat är utvecklad för att hantera program och applikationer inom multimedia som kan kräva mycket. Den har också stöd för full-HD. (Broadcom, 2015) Processorn är utrustad med tre stycken Controllers för SPI, vilket är protokollet som driver bussen på Raspberry Pi. Hårdvarumässigt drivs Raspberry Pi genom att en mikro-USB kabel kopplas till en strömkälla. Till skillnad från en vanlig dator, som traditionellt sett har minne sparat på en hårddisk, så finns det på kortet en läsare för SD-kort, där SD-kortet används som minne för bland annat operativsystemet. På vissa versioner av kortet finns det ännu ett USB-uttag, vilket ökar möjligheten för utbyggnad av mikrodatorn genom att det går att koppla in skärmar, tangentbord och andra tillbehör. (Raspberry Pi, 2015)

5.1.2 Mjukvara

Mikrodatorn är i första hand kompatibel med Linux och det objektorienterade programspråket Python, och användaren kan koppla den till valfri skärm för att ha den fungerande som dator. Ett annat vanligt operativsystem är Raspbian, som är en fristående utveckling från Linux anpassad just för Raspberry Pi. (Upton & Halfacree, 2013) Men på grund av att Raspberry Pi som företag förespråkar open source finns det mängder av officiella och inofficiella operativsystem som går att använda, där vissa är specificerade för att passa ett visst användningsområde. Dock krävs det att man installerar en kompilator för att kunna köra olika program. (Raspberry Pi, 2015)

5.1.3 Påbyggnadsmoduler

Det finns en mängd olika hårdvarulösningar som går att koppla till en Raspberry Pi för att utöka funktionaliteten. Då Raspberry Pi ofta används för projekt inom multimedia är det vanligt med skärmmoduler eller till exempel högtalare. Förutom befintliga shields finns det via internet många forum eller liknande med tutorials för hur man själv bygger till önskad modul. För att även kunna använda shields som är utvecklade specifikt för Arduino finns det specialbyggda kort som är speciellt utvecklade för att sköta anslutningen mellan en Arduino shield till en Raspberry Pi. En sådan överbryggning fungerar genom att den ansluts ovanpå Raspberry Pi och sedan monteras den aktuella shield som användaren önskar ovanpå överbryggningen. (Cooking-Hacks, 2015)

5.2 Arduino

Arduino är en utvecklingsbar mikrocontroller, där tanken från början, precis som med Raspberry Pi, var att erbjuda ett prisvärt och roligt sätt för unga att tillgodogöra sig kunskap kring teknologi och programmering. (Blum, 2013)

5.2.1 Hårdvara

Det finns många olika versioner av det officiella Arduinokortet, där vissa kort är specialdesignade för att passa särskilda ändamål. Arduino Lilypad är ett exempel på detta, då den är speciellt utvecklad för att sys in i kläder och tyger. Det som är de största skillnaderna mellan olika Arduino-versioner är antalet in-/utgångar, minnesstorlek samt hur strömtillförseln sker. (Arduino, 2015)

Genom att via USB koppla Arduinokortet till sin dator går det att överföra programkod till mikrocontrollern, som fungerar som själva hjärnan på kortet. USB-uttaget fungerar även som transport för in- och utsignaler mellan Arduinon och datorn. På kortet finns det analoga och digitala in- och utgångar som kan ställas in efter önskad funktion samt fyra stycken LED-dioder där den ena kan användas för programmerarens eget önskemål och de andra tre har förinställda funktioner såsom att lysa då kortet har aktiv strömtillförsel. Både mjukvaran och hårdvaran till Arduino är open source, det vill säga tillgänglig för alla att modifiera efter eget tycke. (Boxall, 2013)

5.2.2 Mjukvara

Då det är en mikrocontroller används inget specifikt operativsystem för att sköta själva Arduinon, utan den går att programmera från en hemdator via Arduinos egna utvecklingsmiljö (engelska Integrated Development Environment, även förkortat IDE). (Blum, 2013) Programmet laddas ned från Arduinos hemsida och efter installation går det direkt att skriva och överföra kod till Arduinokortet. Själva miljön är programmerad i Java men språket som används för kodning är en modifikation mellan C och C++, där vissa konventioner har förändrats och ibland förenklats. I miljön går det att implementera flera olika bibliotek beroende på vilka funktionaliteter ens program kräver, vilka ständigt är under utveckling. (Arduino, 2015) Utvecklingsmiljön är indelad i tre huvudområden; kommandofält innehållande ikoner som *file, sketch, tools* etc, textredigerare där du skapar ditt program, eller *sketch* som det kallas i Arduinovärlden, samt ett fönster för verifikation och status- eller felmeddelande av programkod. (Wheat, 2011) I mikrocontrollerns flash-minne finns en Bootloader, vilket innebär att det inte krävs någon extern hårdvara för att ladda över ett nytt eller uppdaterat program till mikrocontrollern. Till utvecklingsmiljön finns *Serial Monitor* som det går att skriva ut text till, samt att det går att programmera koden så att det som skrivs i Serial Monitor från användaren läses in av programmet. (Arduino, 2015)

5.2.3 Påbyggnadsmoduler

Till Arduino finns det många varianter av shields som är gjorda för att implementeras med kortet. Genom att addera moduler och komponenter går det att skapa Arduinoplattor med vitt skilda användningsområden och i grundutbudet och på internet finns det flera olika bibliotek som innehåller funktioner för olika sorters hårdvarumoduler. (Boxall, 2013) De flesta shields är byggda för Arduino Uno, som är en av de mest populära versionerna. Till den har påbyggnadsmodulerna anpassats för att det skall vara möjligt att enkelt klicka på modulen ovanpå själva Arduinokortet, och man kan på så sätt bygga på funktionalitet ovanpå vartannat. För att det även skall gå att implementera till andra varianter av Arduino har många tredjepartsutvecklare utvecklat metoder för att speciella shields skall gå att använda till andra kort utan för mycket modifikation. (Arduino, 2015)

5.3 BeagleBone

Texas Instrument är skaparna bakom BeagleBone vars ambition var att skapa en kraftfull, påbyggnadsbar dator till ett överkomligt pris. Det finns två typer av BeagleBone på marknaden, originalet BeagleBone som lanserades november 2011 samt BeagleBone Black som introducerades i början på 2013. Mikrodatorn, som storleksmässigt kan jämföras med ett kreditkort, är till för hobbyister likväl som bruk i utbildningssyfte och skaparnas strävan är att användarna skall kunna skapa innovativa projekt och dela med sig av ny utveckling. Alla hårdvarudetaljer finns således tillgängliga via skaparnas hemsida och plattformen har öppen källkod. (Barret and Kridner, 2013)

5.3.1 Hårdvara

BeagleBone har inbyggd Ethernet-funktion på kortet, vilket möjliggör internetkommunikation. Den är även utrustad med en Arm-processor samt 2x46 in-/utgångar och kan drivas via USB, men har också möjlighet att drivas från en extern strömkälla. Varianten BeagleBone Black är en vidareutveckling med bland annat kraftigare processor, låg energiförbrukning och ett HDMI-uttag för *video output*. (Beagleboard, 2015)

5.3.2 Mjukvara

Mikrodatorn kan användas av dem som vill skapa alltifrån enklare projekt till mer komplexa system. Mikrodatorn kommer förinstallerad med operativsystemet Linux, vilket är utvecklarnas rekommendation att använda för att kunna utnyttja både kraft och kapacitet till fullo. BeagleBone kan dock användas av både Linux, MS Windows eller Mac OS X användare på grund av BoneScript, som är en användarvänlig programmeringsmiljö som kan nås via användarens webbläsare. BoneScript består av ett JavaScript-bibliotek och nås via webbservern ty BoneScript finns förinstallerat på BeagleBone-kortet. (Barret and Kridner, 2013)

5.3.3 Påbyggnadsmoduler

Moduler designade för att implementeras med BeagleBone kallas capes, vilket kan jämföras med shields för Arduino samt Raspberry Pi. Det finns både capes som är designade av företaget samt mindre officiella versioner och varianter. En cape kan vara alltifrån ett kretskort utrustat med HDMI-uppkoppling och en LED-skärm till en färdigdesignad väderstation som bara är att koppla in till sin BeagleBone. (Beagleboard, 2014)

5.4 Pughmatris och urval huvudmodul

För att på ett systematiskt sätt välja ut vilken huvudmodul som skulle användas konstruerades en Pughmatris, där BeagleBone sattes som referenslösning och de andra två därefter jämfördes med referensen, där ”+” betyder att modulen anses ha bättre förutsättningar och ”-” betyder att den anses ha sämre förutsättningar. ”0” sätts om förutsättningarna anses neutrala mot referensen.

Chalmers	**Pughmatris (Relativ beslutsmatris):**		
Utfärdare: Cecilia Hernqvist Stina Jange			***Skapad:*** 2015-04-17 ***Modifierad:*** 2015-04-22
Kriterier	**Alternativ**		
	BeagleBone (Ref)	**Arduino**	**Raspberry Pi**
Programmeringsspråk	0	+	0
Antal in-/utgångar	0	-	-
Tillgänglighet av påbyggnadsmoduler	0	0	0
Möjlighet till CAN-kommunikation	0	0	0
Användarvänlighet	0	+	-
Kunskap på företaget	0	+	+
Tillgänglig information/hjälpmedel	0	0	0
Projektets syfte	0	+	0
Antal +		4	1
Antal 0		3	5
Antal -		1	2
Nettovärde	0	3	-1
Rangordning	2	1	3
Vidareutveckling	NEJ	JA	NEJ

Figur 5.1 - Matris för urval av hårdvara

Den huvudkomponent som utifrån Pughmatrisen valdes för det specifika projektet var Arduino. Uppgiften hade gått att utföra med alla tre komponenterna men ett av argumenten emot BeagleBone och Raspberry Pi var att uppgiftens natur inte kräver att huvudmodulen skall använda sig av ett eget operativsystem. Arduino ansågs därmed mest lämplig i egenskap av att den fungerar mer som en ren mikrocontroller, vilket är önskvärt i inbyggda system och reglerkretsar för att styra elektriska komponenter. Andra argument var att Arduino går att programmera i C, vilket visserligen även fungerar för BeagleBone och Raspberry Pi men för de två är C-kod inte standardvalet. Tillgänglighet av påbyggnadsmoduler ansågs vara ungefär likvärdigt, då alla tre är open source och det finns mycket tillgänglig hårdvara att få tag i. Även möjlighet till CAN-kommunikation ansågs likvärdig då det till alla tre finns påbyggnadsmoduler med den specifika uppgiften. Även personalens kunskap på Consat spelade in, och då fler verkade ha erfarenhet av Arduino och Raspberry Pi än av BeagleBone togs detta i beaktning för att underlätta framtida möjligheter att få hjälp och råd av handledare.

Vidare efter beslut att använda huvudmodulen Arduino valdes varianten Arduino Mega 2560 som på kortet har mikrocontrollern ATmega2560. Arduino Mega 2560 är en kraftfullare variant av den traditionella Arduino Uno. Arduino Mega 2560 innehåller större minne samt 54 st digitala pins samt 16 analoga ingångar med 10-bitars A/D-omvandlare. (Arduino, 2015) Den ansågs lämplig för att möjliggöra för Consat att vidareutveckla modulen med fler funktioner efter projektets slut.

5.5 Urval övrig hårdvara

När väl huvudkomponenten valts fortgick urval av ytterligare hårdvara. Efter research och i samråd med huvudhandledaren på Consat beställdes en CAN-shield, två relä shield’s samt två

kopplingsbrädor. Övrig hårdvara såsom lysdioder, motstånd, kopplingskablar fanns att tillgå i laborationsrummet på Consat.

Den CAN-shield som används i detta projekt har en DE-9 kontakt (en elektrisk kontakt som ofta används i datorsammanhang) som kan kommunicera med en bils styrenhet då en OBD-II kabel ansluts mellan CAN-shielden och bilen. I detta projekt kommer CAN-kommunikationen att ske mellan en dator och en CAN-shield, som i sin tur kommunicerar med Arduinon. På så vis kommer Arduinon ta emot meddelanden via CAN-bussen som säger vad som skall utföras. Den CAN-shield som är aktuell i detta projekt är utrustad med Microchip MCP2515 CAN-Controller samt MCP2551 CAN-Transciever. (Sparkfun, 2015)

De relä shields eller reläkort som beställdes är färdigbyggda samt testade och styrs från Arduinon. Reläkorten är vardera utrustade med åtta stycken reläer som styrs med 5 V. På reläkortet finns nio lysdioder, där en röd lysdiod indikerar huruvida reläkortet är på och resterande åtta gröna lysdioder indikerar varje reläs status, där tänd diod innebär att det aktuella reläet är aktiverat. Det finns tre utgångar på vardera relä, COM, NC och NO, där respektive står för gemensam, normalt stängd samt normalt öppen. Då ett relä aktiveras, dvs. då NC-kontakten bryts och NO-kontakten sluts, aktiveras lysdioden för aktuellt relä.

De kopplingsbrädor som beställdes är vardera utrustade med två skenor för matningsspänning respektive jord, 10 kolumner och sammanlagt 63 rader.

I senare skede av projektet beställdes en spänningsomvandlare, för att möjliggöra att Arduinon får sin matningsspänning via ett av de externa nätaggregaten istället för via USB-kabeln från datorn. Detta kommer att implementeras i slutskedet när programkoden är klar då Arduinon inte kommer att kräva USB-uppkoppling längre. I samma veva beställdes även två stycken D-subkontakter, en hane och en hona, som kommer utgöra gränssnittet för de pin-paren som skall kunna kortslutas.

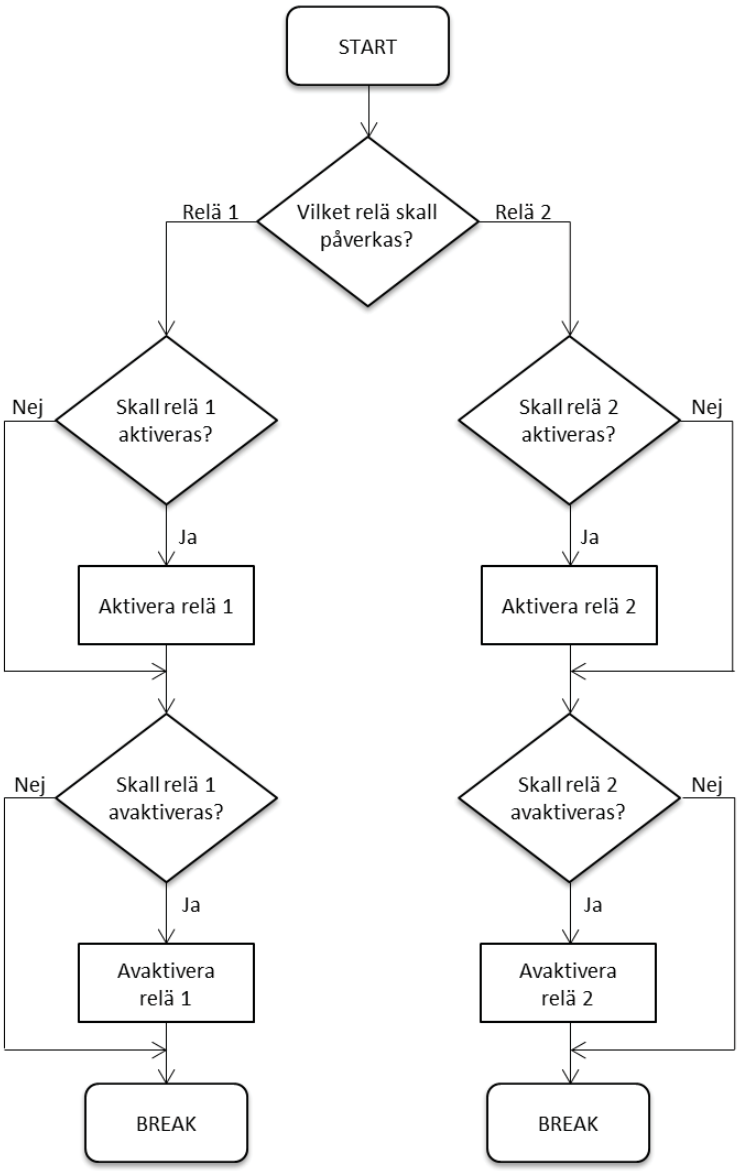
6 Utveckling av kretskopplingar

Efter leverans av hårdvara var första steget att göra en enkel uppkoppling av Arduinokortet, relä shield, motstånd, lysdioder samt kopplingsbräda för att bli bekanta både med hårdvara samt Arduinos utvecklingsmiljö. En enklare sketch skrevs innehållande funktioner för att aktivera samt avaktivera två reläer utefter en viss sekvens, där reläerna hade kopplats samman med var sin lysdiod. Då ovan nämnda koppling och sketch visat sig fungera utvecklades programkoden vidare, där målet var att kunna styra aktivering respektive avaktivering av reläerna via utvecklingsmiljön. Detta genomfördes genom att tillägna olika kommandon i programmet varsitt värde, där aktivering av kommandot gjordes genom att skriva in de olika värdena i Serial Monitor. När programmet sedan tolkade in värdet utfördes den funktion som värdet innebar, såsom att aktivera eller avaktivera ett relä.

Vidare upptäcktes att fler hårdvarukomponenter krävdes för att kunna fortsätta med uppkoppling av Arduinon och komma igång med utförandet av steg 1. Därmed gjordes efter överläggning med huvudhandledare en ny beställning innehållande D-subkontakter samt stackningsbara hylslister, som används för att enkelt koppla samman Arduinokortet med CAN-shielden. Hylslisterna har 8-poliga respektive 6-poliga kontakter vilket stämmer överens med in-/utgångarna på CAN-shielden och D-subkontakten 37 poler, där polerna (två och två) utgör de pin-par som skall kunna kortslutas. Genom att ansluta motsvarande D-subkontakt som är inkopplad till kretskortet som skall testas går det således att kortsluta valda trådpar på kretskortet.

6.1 Steg 1 – Spänning on/off

Steg 1 inleddes med att rita upp ett flödesschema (se Figur 6.1) för att få en bild av hur processen skulle se ut. Detta underlättade vid utveckling av programkod längre fram då det gav en tydligare överblick av vilka steg det var som skall göras och vad som skall utföras. Inledningsvis användes ingen CAN-kommunikation in till Arduinon utan styrsignalerna sköttes via Serial Monitor i utvecklingsmiljön. Till att börja med utfördes programmet genom att läsa in vilket relä som skulle påverkas, alltså vilken av de två matningsspänningarna det är som skall kontrolleras. Därefter sker tolkning huruvida reläet skall aktiveras eller avaktiveras, alltså om spänningen skall matas igenom eller brytas. Därefter hoppade programmet ur funktionen och inväntade nästa kommando.



Figur 6.1 - Flödesschema steg 1

Det första som gjordes var att använda den redan uppsatta testkopplingen med två reläer som vardera var kopplade till en lysdiod, där USB-kabeln in till Arduinon stod för strömförsörjningen och dioderna ger visuell återkoppling om vilket relä som är aktiverat. För att få detta att fungera användes en switch-sats i programkoden där fyra olika case kunde utföras, beroende på vilket värde som skrevs in i Serial Monitor, där de fyra casen är aktivering och avaktivering av de båda reläerna. Reläerna fungerar genom att styrsignal ges via respektive digital pin på Arduinon, där styrsignalen aktiverar reläspolen och på så sätt sluts reläet så att lysdioden tänds.

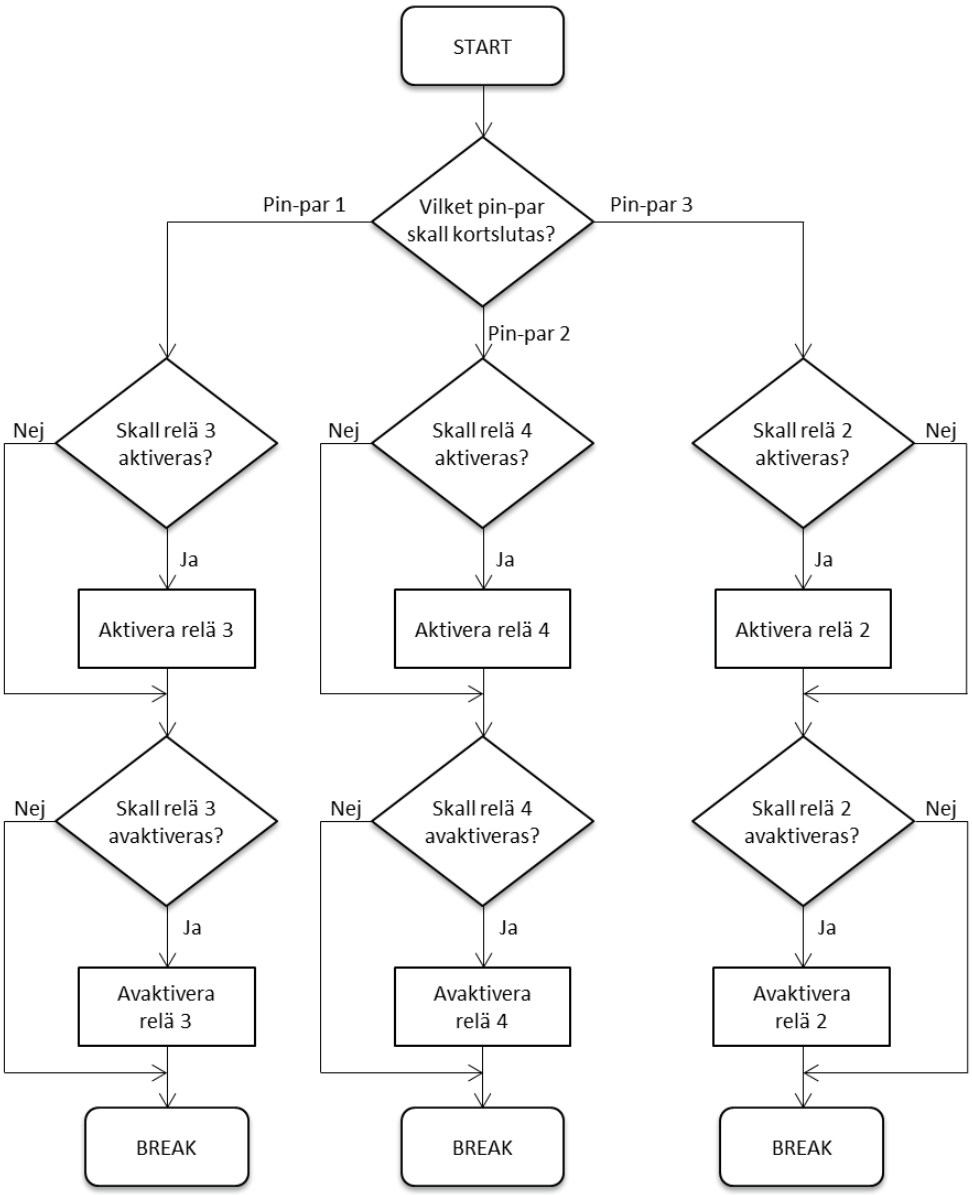
När detta fungerade tillfredsställande utvecklades kopplingen. Ett spänningsaggregat adderades tillsammans med spänningsregulatorn varpå uppkopplingen ändrades genom att koppla nätaggregatet till en av kopplingsplattorna. Matningsspänningen från aggregatet, via spänningsregulatorn, skall fungera som en möjlig spänningskälla till Arduinon, som kräver reglering av spänningen då den bara klarar matningsspänning på 12 V. Det andra användningsområdet för aggregatet är det tilltänkta i steg 1, att vara inkopplad till det aktuella reläet, där spänningen ut till kretskortet skall kunna vara högre än 5 V. På så sätt kunde

styrsignal ges från Arduinon att aktivera reläet, vilket medförde att spänning från nätaggregatet tände lysdioden. Därefter löddes kablar på två hon-banankontakter, vilka skall användas för att ansluta det externa kretskortet som skall testas med modulen. Dessa kopplades in på kopplingsbrädan på samma rad som pluspolen till lysdioden, så att spänning når banankontakterna då reläet är tillslaget. Därmed går det att styra om det externa kretskortet blir försörjt med spänning eller ej. Senare adderades ett till par banankontakter för att möjliggöra anslutning av två spänningsaggregat, som var tanken med detta steg.

Programkoden för steg 1 kom att modifieras under projektets gång, där switch-satsen ersattes med olika if-satser. Detta innebar inga egentliga förändringar av flödesschemat då programkoden utförde samma funktioner som tidigare, dvs. if-satserna aktiverar respektive avaktiverar önskat relä beroende på vilken styrsignal som ges. Ändringarna utfördes med anledningen att underlätta CAN-kommunikationen, främst återmatning av status, som beskrivs mer ingående i Kapitel 7.

6.2 Steg 2 – Generera kortslutning

Även denna utförandefas inleddes med att rita upp ett flödesschema för att få en övergripande bild av hur programkoden skulle kunna utformas (se Figur 6.2). Flödesschemat ritades initialt upp för möjlig kortslutning av tre stycken pin-par, men funktionaliteten utökades senare till kortslutning av fler pin-par än så.



Figur 6.2 - Flödesschema steg 2, illustrerat med tre reläer

Uppkopplingen av hårdvara för steg 2 var relativt lik uppkopplingen för steg 1. Inledningsvis användes inte heller här någon CAN-kommunikation, utan styrsignalerna till Arduinon sköttes via Serial Monitor i utvecklingsmiljön. Serial Monitor läser in vilken funktion som skall aktiveras, dvs. vilket relä på relämodulen som skall påverkas. Till en början var reläerna kopplade från COM till matningsspänning samt från NO, via ett motstånd, till en lysdiod på kopplingsbrädan. Detta ändrades i ett senare skede, så att varje relä kopplades från COM samt NO till de pin-par på D-subkontakten som önskas kortslutas. I början användes endast tre reläer på relämodulen där reläerna, beroende på vilken digital pinne på Arduinon som sätts till hög, genererar varsin kortslutning. Antalet reläer för steg 2 utökades till 14 stycken i slutet av projektet, detta för att möjliggöra kortslutning av ytterligare pin-par.

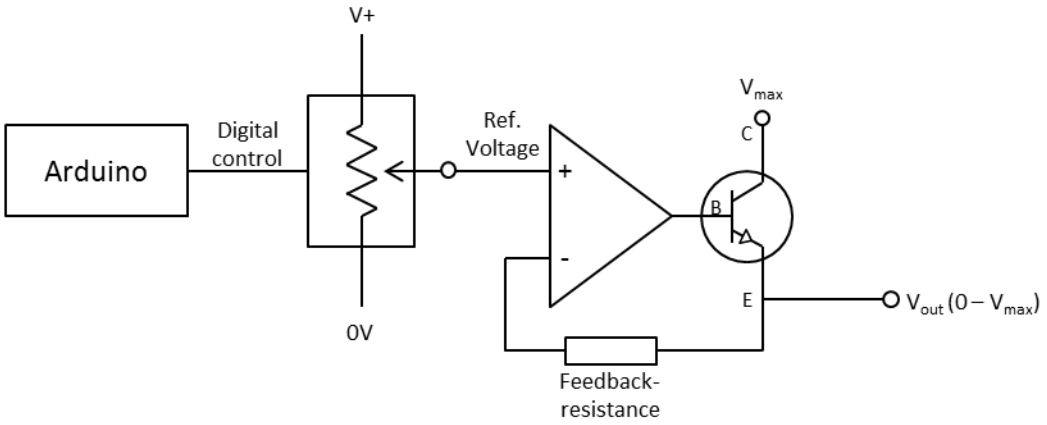
Inledningsvis användes även i denna programkod en switch-sats, som kontrollerades via Serial Monitor, innehållandes ett case för varje relä som skulle påverkas. Av samma anledning som angetts i Avsnitt 6.1 ersattes switch-satsen med if-satser.

6.3 Steg 3 – Variera spänning

Steg 3 var från projektets start satt som ett önskemål som skulle utföras i mån av tid. På grund av bristen på tid kunde detta steg ej utföras, men för att underlätta vidare utveckling av modulen togs tre stycken teoretiska tillvägagångssätt fram över hur steg 3 eventuellt skulle kunna se ut. Vid jämförande av de tre förslagen kommer aspekter som tid och hållbarhet att beaktas.

6.3.1 Spänningsvariation förslag 1

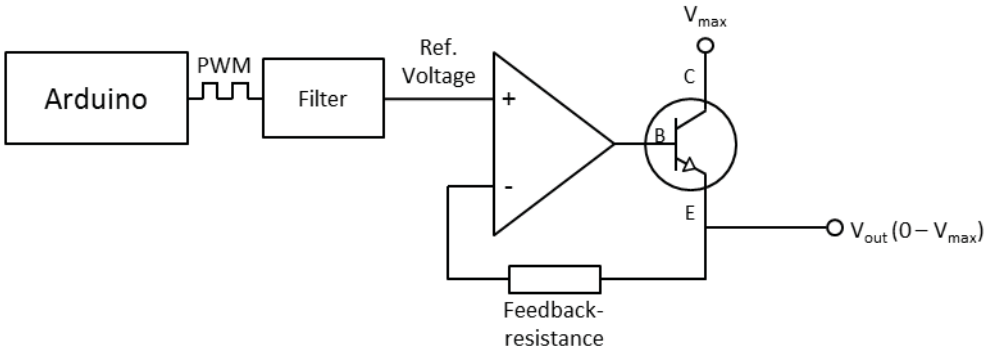
Figur 6.3 visar ett blockschema över hur det första förslaget på spänningsvariation ut till kretskortet skulle kunna realiseras. Figuren visar endast en överblick, och vid realisering skulle flertalet kringkomponenter krävas. Vad det första förslaget innebär är att Arduinon är programmerad till att styra en digital potentiometer via en styrsignal som sedan, beroende på värdet på styrsignalen, anpassar utsignalen från potentiometern och som i sin tur blir referensspänning till resten av kretsen. Referenssignalen går därefter in i en operationsförstärkare där dess förstärkning, *gain*, förstärker signalen. OP-förstärkaren är sammankopplad med en transistor som jämför det nya värdet på signalen och skickar ut V_{out} som motsvarar det önskade, förstärkta värdet. Via transistorn sker också en återmatning till minuspolen på OP-förstärkaren, som tillsammans med pluspolen bildar en differens som är den som förstärks.



Figur 6.3 - Krets förslag 1

6.3.2 Spänningsvariation förslag 2

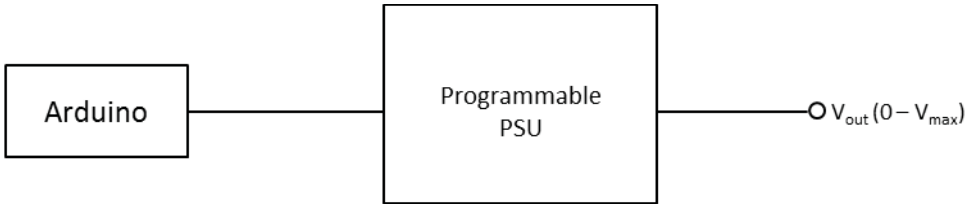
Precis som i förslag 1 används även i förslag 2 en OP-förstärkare med återmatning samt en transistor för att få in en referensspänning som sedan förstärks och jämförs med maxvärdet på spänningen. Det som skiljer från förslag 1, vilket också visas i Figur 6.4, är att förslag 2 använder sig av PWM istället för en potentiometer. Då Arduino Mega 2560 är utrustad med flera PWM-utgångar skulle en av dessa kunna vara dedikerad för ändamålet. Det skulle innebära att beroende på vilket värde som ställs in så anpassar PWM-signalen pulserna till att rätt värde skickas ut, där PWM-signalen sedan passerar ett filter för att stabilisera signalen och filtrera bort eventuellt brus. Därefter skickas signalen in som referens i resterande del av kretsen.



Figur 6.4 - Krets förslag 2

6.3.3 Spänningsvariation förslag 3

Det tredje förslaget som visas i figur 6.5 handlar om att använda ett programmerbart nätaggregat. Till skillnad från ett vanligt nätaggregat, där önskvärd output oftast ställs in för hand på aggregatet, kan ett programmerbart nätaggregat ställas in via signaler från ett styrsystem. I detta fall skulle då Arduinon programmeras till att kunna styra aggregatet till att skicka ut önskad output till kretskortet som skall testas.



Figur 6.5 - Krets förslag 3

6.3.4 Jämförelse förslag

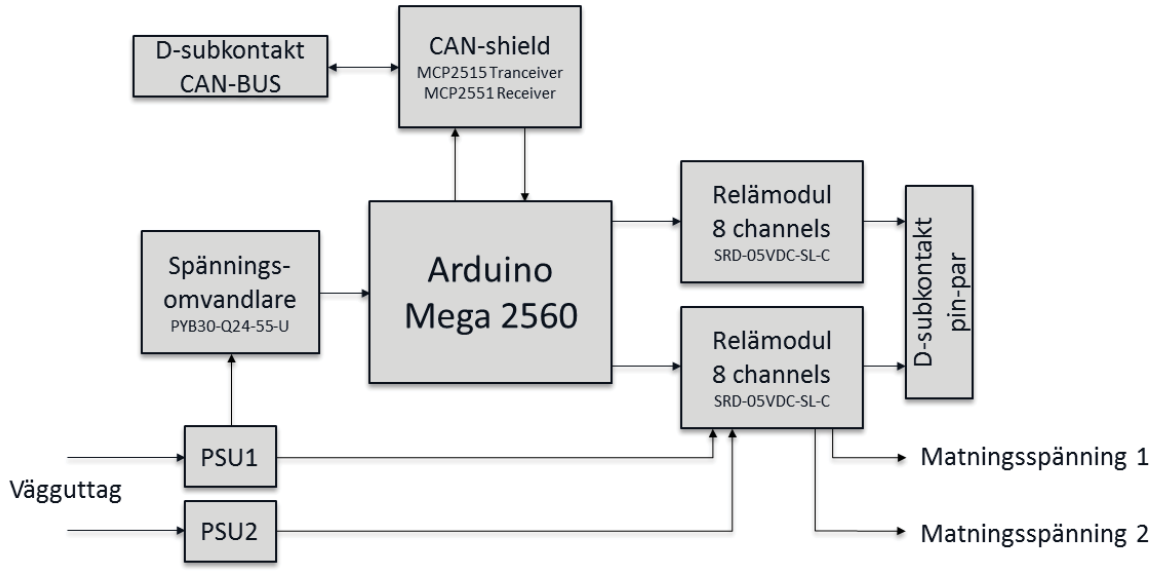
Det som talar emot de två första förslagen är att det blir mer ren hårdvarukoppling med små enskilda komponenter, där det även kommer att krävas kringkomponenter för att få kretsen att bli stabil. Det som även bör övervägas är hållbarheten på lösningen, vilket också talar för förslag 3 i och med att ett sådant nätaggregat är fabrikstillverkat och innefattar alla hårdvarukomponenter som krävs. Vid förslag 1 och 2 krävs det mer förundersökning kring exakta egenskaper på respektive komponent som krävs, samt hur dessa interagerar med varandra, vilket kopplar in tidsaspekten då det kan vara tidskrävande. Att det i förslag 3 är en befintlig modul underlättar även felsökning samt eventuella reparationer då det finns dokumentation kring modulen och ett företag som distribuerar produkten. Det som istället talar för förslag 1 eller förslag 2 är anpassningsmöjligheterna kring att bygga upp och forma kretsen på det sätt det aktuella projektet kräver, samt att det öppnar möjligheten för att använda flera olika sorters nätaggregat som finns på marknaden, både ställbara och fasta. Dock bör diskussion ske kring valet mellan att använda PWM eller en digital potentiometer samt huruvida de skilda sätt som dessa två komponenterna dimensionerar spänningen på kan påverka kretsen på något sätt, eftersom potentiometern lägger sig på ett fast värde mellan två gränsvärden medan PWM använder pulser som pendlar mellan 0 V och V_{max}. Förutom det övervägandet har förslag 2 fördelen att man använder sig av befintliga PWM-utgångar som är uppsatta på Arduinon.

6.4 Kretskoppling och blockschema

Då steg 1 och steg 2 var utförda ritades ett blockschema upp (se Figur 6.6) för att övergripligt visa hur samtliga hårdvarukomponenter är kopplade till varandra. Vid sammansättning av den slutliga produkten monterades hårdvaran på en aluminiumplatta, där dittills provisoriska trådar byttes ut mot permanenta som löddes fast (se Bilaga 6). I detta skede monterades även samtliga kontakter som behövs för att modulen skall vara klar att använda. Arduino Mega 2560 utgör själva styrenheten som matas med spänning från ett av nätaggregaten (PSU) via spänningsomvandlaren. Den omvandlade matningsspänningen är även kopplad till V_{CC} på de två relämodulerna och från jord på spänningsomvandlaren till relämodulernas jordkontakter.

De två nätaggregaten är anslutna till varsitt relä på den ena relämodulen, där kopplingen till reläet ser likadan ut för de två nätaggregaten. Matningsspänning går till COM på aktuellt relä och från NO-kontakten går en banankontakt som kommer generera en spänning ut till kretskortet då reläet sluts. Från jord på spänningsomvandlaren sitter den andra banankontakten som skall kopplas till kretskortet. De resterande 14 reläerna är kopplade till den D-subkontakt som utgör gränssnitt för pin-paren som skall gå att kortsluta (se Bilaga 3), varför endast 28 stycken av de totalt 37 polerna på kontakten används. Varje pin-par går till COM och NO på respektive relä och då reläet sluts erhålls kortslutning.

Vidare är Arduinon sammankopplad med CAN-shielden som i sin tur är kopplad till en dator, för att möjliggöra kommunikation enheterna emellan. För att ge en mer detaljerad insyn i uppkopplingen av hårdvaran ritades ett kretsschema i programmet KiCad (se Bilaga 5).



Figur 6.6 - Blockschema hårdvara

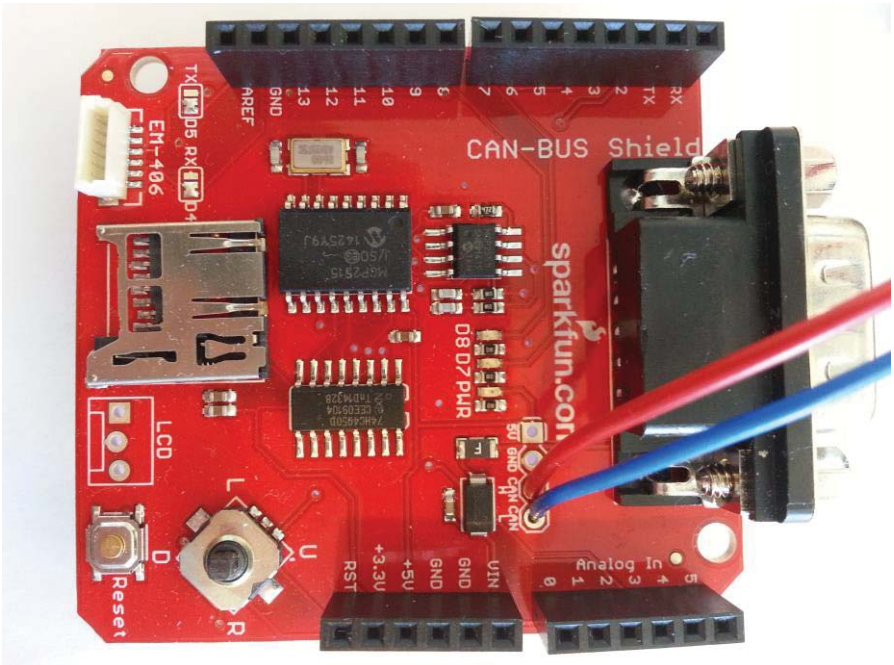
7 Programkodutveckling

Detta kapitel behandlar den programkod som utvecklats samt tillämpats under projektets gång. När varje enskilt program av de två stegen fungerade korrekt var för sig så var det dags att sammanställa dessa. Ett huvudprogram utvecklades i Arduinos utvecklingsmiljö för att kunna utföra de funktioner som redogörs i Kapitel 6, och varje sekvens testades återigen för att säkerställa att ingenting gick fel eller att någon funktionalitet störde någon annan. I detta steg lades det även till en sats som väntade på besked ifrån användare angående vilket av testerna som skulle utföras, där varje test till en början gavs ett specifikt nummer. Detta medförde att den siffran kunde skrivas in i Serial Monitor och på så sätt tolkade programmet vilket test som skulle utföras. Därefter inleddes informationssökning angående CAN-bibliotek för CAN-shields kompatibla med Arduino. Nedan beskrivs de olika moment som behandlats under programkodutvecklingen.

7.1 CAN-kommunikation

För att kunna kommunicera via CAN hämtades ett CAN-bibliotek hem från Github som är en hemsida innehållande öppna källkoder. Gruppen gjorde flertalet informationssökningar angående vilket bibliotek som var mest lämpat för den CAN-shield som används i detta projekt. Under faktainsamlingen uppdagades att alla bibliotekskoder var kompatibla med Arduino Uno och att det krävdes modifiering för att möjliggöra användning för Arduino Mega 2560. Efter granskning av datablad för Arduino Uno (Arduino Uno Reference Design, datablad) och Arduino Mega (Arduino Mega 2560, datablad) stod det klart vilka ändringar som behövde göras i koden. Arduino Uno använder digitala pinnarna D10, D11, D12 och D13 för SS, MOSI, MISO och SCK för SPI-bussen medan Arduino Mega 2560 använder digitala pinnarna D53, D51, D50 och D52. (Se Bilaga 7). CAN-shielden som tillämpas i projektet är kompatibel med olika typer av Arduinokort, dock krävdes modifiering då även shielden var anpassad för montering ovanpå Arduino Uno, dvs. använder samma SPI-pins som var angivna i CAN-biblioteket. Genom att kapa benen på hylslisterna för pin D10, D11, D12, D13 på CAN-shielden och dra kopplingstrådar till motsvarande pins D53, D51, D50 och D52 på Arduino Mega 2560 skapades kontakt mellan de två korten och shielden kunde efter det monteras (Se Bilaga 4).

Datablad över CAN-shielden visade att D-subkontakten som finns placerad på modulen var av annan struktur än vad som är standard. Vanligtvis brukar pin 2 och 7 gå till CAN *low* respektive CAN *high*, men på den CAN-shield som används i detta projekt går pin 5 och 3 till CAN *low* respektive CAN *high*. Problemet åtgärdades genom att inte använda den befintliga kontakten på kortet utan löda fast kablar från vardera CAN *low* och CAN *high* (som fanns tillgängliga på CAN-shielden och som visas i Figur 7.1) till en D-sub 9-pols honkontakt. (Sparkfun, datablad, 2010)



Figur 7.1 – Författarnas egen bild. CAN-shield med trådar lödade till CAN high samt CAN low.

CAN-shielden kopplades sedan in till datorns USB-uttag via en kabel som i motsvarande ände har en D-sub 9-pols hane. I samråd med personal på Consat placerades två stycken 120 ohms motstånd, ett i vardera ände av gränssnittet mellan CAN-shielden och USB-kabeln, vilket görs för att undvika störningar på CAN-bussen.

Med hjälp av programmet Busmaster kunde CAN-busskommunikationen testas. Då det till en början endast skrevs ut felmeddelanden i Busmaster konsulterades handledarna på Consat, varpå gruppen fick hjälp med att felsöka bibliotekskoden samt undersöka huruvida SPI-linjerna sände ut meddelanden via CAN-bussen. Genom att med ett oscilloskop mäta om det var någon spänningsvariation på bussen upptäcktes att det inte skickades ut några meddelanden via CAN-shielden.

Till en början fann gruppen endast CAN-bibliotek som var utförda för SeeedStudio CAN-BUS shield. Denna CAN-shield har samma controller och transceiver som Sparkfuns CAN-shield, som används i projektet, varvid gruppen ansåg denna kod som lämpad. Efter flertalet försök med diverse CAN-bibliotek utan framgång rådfrågades handledarna på Consat. Ett CAN-bibliotek utvecklat för den CAN-shield som gruppen använde sig av hämtades hem, även denna gång från hemsidan Github. (Github, SK Pang Electronics, 2015) Gruppen hade tidigare försökt få igång CAN-kommunikationen med hjälp av detta bibliotek, men då det innehöll ytterligare funktioner för SD-kort, LCD-display etc. hade gruppen inte lyckats förändra samt utveckla koden korrekt. Efter flertalet modifikationer kunde koden kompileras och laddas upp felfritt. Dock kvarstod samma problem med CAN-kommunikationen varpå den tekniska handledaren rådde gruppen att göra ett förbindelsetest med summer med en multimeter. Funktionen med summer kan exempelvis påvisa huruvida en kabel eller en lampa är hel samt om en säkring gått. Gruppen kunde utifrån förbindelsetestet undersöka om trådarna som kopplats via CAN-shielden och Arduino Mega 2560 var hela, om kortslutning orsakats vid lödning, samt ifall MCP2515 Controllern och MCP2551 transceivern hade kontakt med SPI-pinnarna. Det visade sig vara kallödning mellan D11 samt D13 och MCP2551 på CAN-modulen. Detta åtgärdades genom att bättra på lödningen på D11 och

D13. Efter kompilering samt uppladdning av CAN-biblioteket kunde kommunikation på CAN-bussen ske.

Nästa steg var att kunna styra reläerna genom att skicka ut meddelande-ID ut på bussen via Busmaster. Till att börja med användes makrodefinitioner med adresser vilka användes i include-filen Canbus.cpp i en switch-sats där delar av den ursprungliga programkoden för steg 1 samt steg 2 implementerades (Se Bilaga 15). Ytterligare makrodefinitioner krävdes då fler case-satser användes än vad som ursprungligen fanns i biblioteket, och därmed definierades fler adresser som sedan tillämpades i respektive case-sats. Genom att lägga till de adresser som valts i Busmaster under *TX Message List* och sedan skicka ut meddelande för vald adress kunde reläerna styras. Dock ändrades programmets funktionalitet i ett senare skede efter önskemål från Consat, och istället för en switch-sats som inväntar kommando för respektive funktionalitet implementerades programkod som kan utföra flera kommandon utifrån ett CAN-meddelande, för att möjliggöra styrning av flera reläer med ett kommando. Därmed tillämpades inte längre de makrodefinitioner med adresser som tidigare användes, utan detta ändrades till att endast använda en adress för meddelanden in till Arduinon och en adress för meddelanden ut från Arduinon. (Se Bilaga 1) Genom att skicka ett meddelande som innehåller all information kring vilka reläer som skall aktiveras samt avaktiveras till Arduinon kan flera kommandon utföras under samma programsekvens. Den andra adressen behandlar sedan ett meddelande genererat av Arduinon som skickar tillbaka aktuell status på reläerna.

7.2 Styrprogram

Efter ändringarna kring hur programmet skulle utföras togs färdig programkod fram, där huvuddelen av den skrivna koden ligger i include-filen Canbus.cpp (Se Bilaga 13). Nedan följer beskrivning kring den skrivna programkoden där de delar som berör sändning och mottagning av meddelande via adresserna ingår. I bilaga 13 finns programkoden i sin helhet, men ett utdrag från programkoden kan ses i figur 7.2, figur 7.3 och figur 7.4.

Vid körning av programmet initieras först CAN-kommunikationen i en underliggande fil i biblioteket, där kontroll sker bland annat kring hastigheten på bussen samt om noderna i nätet har kontakt. Då programmet testkörs via Arduinos utvecklingsmiljö ges en utskrift i Serial Monitor med texten ”Can init ok” ifall initieringen av kommunikationen fungerar korrekt.

Programmet är uppbyggt genom att vissa bytes i meddelandet används genom att modifiera hur bitarna i varje byte är ställda (Se Bilaga 2). En byte består av åtta stycken bit, där varje relä som skall styras är tilldelad en bit. Byte 0 i meddelandet är tilldelad steg 1, och eftersom det i projektet endast anpassas till användningen av två stycken nätaggregat så används endast bit 7 och 6. Nästkommande tre bytes, byte 1, 2 och 3 är tilldelade steg 2, där byte 1 och större delen av byte 2 används för de 14 reläer som är implementerade i det steget, och resterande använda bitar är förberedda för framtida implementering. Kod är skriven för totalt 26 stycken reläer, men det går enkelt att utvidga koden om detta skulle önskas. Varje reläs tilldelade bit kan antingen vara ettställd eller nollställd, där ettställd innebär att reläet skall aktiveras och nollställd betyder att det skall avaktiveras.

För att generera önskat meddelande i Busmaster ändras respektive databyte till ett hexadecimalt värde som motsvarar det åttabitars binära värdet som bildas beroende på vilka bitar man önskar ettställa respektive nollställa. Detta får till följd att varje relä motsvaras av ett visst bitmönster, exempelvis 1000 0000 vilket motsvarar 80 hexadecimalt. När meddelandet är inställt sänder man det från Busmaster, och programkoden hos Arduinon läser därefter av meddelandets identifierare. Ifall det är rätt adress tas meddelandet emot och

initierar programmet, vilket innebär att programmet går in i ett antal if-satser där det görs en koll på respektive bit i varje byte huruvida den är ettställd. Detta görs genom att använda bitvis AND som maskar bort resterande bitar förutom exakt den som den aktuella if-satsen hanterar. Om argumentet visar sig vara sant, det vill säga om den aktuella biten är skild från noll aktiveras reläet. Om maskningen visar att biten istället är nollställd går programmet vidare till tillhörande else-sats, där reläet avaktiveras. Detta förfarande upprepas för samtliga reläer. I figur 7.2 visas denna programsekvens för steg 1. Programsekvensen för steg 2 finns redovisad i bilaga 7.

```
     char CanbusClass::ecu_req(char *buffer)
{
    tCAN message;

    //Check for received frames
    if (mcp2515_check_message())
    {
        //Get data (&-sign means get the address for 'message'
        if (mcp2515_get_message(&message))  )
        {
            //Do only if message identifier equals RECEIVE_FRAME_ID
            if(RECEIVE_FRAME_ID == message.id)
            {

                //********************STEP 1******************//

                //If bitwise AND resolves to true
                if(0 != (message.data[0] & 0x80))
                {
                    //Sets power supply 1 to on
                    digitalWrite(RELAY1, HIGH);
                    Serial.print("Power supply 1 activated. \n");
                }
                //If bitwise AND resolves to false
                else
                {
                //Sets power supply 1 to off
                digitalWrite(RELAY1, LOW);
                Serial.print("Power supply 1 deactivated. \n");
                }

                //If bitwise AND resolves to true
                if(0 != (message.data[0] & 0x40))
                {
                    //Sets power supply 2 to on
                    digitalWrite(RELAY2, HIGH);
                    Serial.print("Power supply 2 activated. \n");
                }
                //If bitwise AND resolves to false
                else
                {
                    //Sets power supply 2 to off
                    digitalWrite(RELAY2, LOW);
                    Serial.print("Power supply 2 deactivated. \n");
                }
            }
```

Figur 7.2 – Utdrag från huvudprogram, med if-satser som styr respektive relä

Efter att programmet har gått igenom vilka reläer som skall aktiveras respektive avaktiveras skickas ett statusmeddelande tillbaka som visar status för samtliga reläer. I figur 7.3 visas inläsning till detta meddelande för steg 1. Det som händer är att kommandot *digitalRead* läser av den digitala pin som är kopplad till ett specifikt relä, där inläsningen återmatar en etta om pinnen är aktiverad och en nolla om pinnen är avaktiverad. Det inlästa värdet maskas sedan genom en bitvis AND-operation som jämför mot det hexadecimala värdet 01, och därefter

skiftas just det värdet, ettan eller nollan, ett visst antal steg för att hamna på den plats i byten där initieringen till det specifika reläet skedde från början.

```
//Bitwise mask and shift left to put bit in the right position
    message.data[0] |= ((digitalRead(RELAY1) & 0x01) << 7);
    message.data[0] |= ((digitalRead(RELAY2) & 0x01) << 6);
```

Figur 7.3 – Utdrag från huvudprogram, avläsning av status på reläer

Efter att samtliga reläer är avlästa sätts meddelandets identifierare till en ny adress, som är den adress som hanterar meddelanden som skall skickas ut från Arduinon. Därefter skickas hela meddelandet tillbaka och det är då möjligt att läsa av meddelandet i Busmaster för att se att reläernas status är korrekta.

```
//Set message identifier to SEND_FRAME_ID
message.id = SEND_FRAME_ID;
//Returns message with information about pin status
mcp2515_send_message(&message);
```

Figur 7.4 – Utdrag från huvudprogram, med statusmeddelande som returneras

8 Resultat

Projektet resulterade i en fungerande modul som kan utföra vissa hårdvarutester på en inkopplad elektrisk styrenhet som skall sitta i ett fordon. Hårdvarutesterna har tidigare utförts manuellt på företaget. Efter avslutat arbete klarar nu modulen av att utföra de obligatoriska tester som finns specificerade i *Precisering av frågeställning* i Kapitel 1, där en utvald styrkrets har programmerats samt hårdvara har valts ut för att kunna utföra uppgiften. Som finns beskrivet i *Syfte* har arbetsgången genomsyrats av tanken att modulen skall ha stor utvecklingspotential och vara kompatibel med flera olika sorters kretskort utan att behöva genomgå alltför omständig modifikation. De steg som var satta som krav, spänning av/på samt kortslutning av pin-par, hann genomföras inom projekttiden men steg 3 fullföljdes tyvärr inte på grund av tidsbrist, som var satt som önskemål där målet var att kunna variera spänningen ut till kretskortet. Under projektet har det dock gjorts en inledande utredning till hur variation av spänning ut till kretskortet skulle kunna utföras. Tre olika lösningsförslag beskrivs i Kapitel 6.3, med syftet att underlätta vidareutveckling av produkten.

CAN-kommunikationen använder sig av två stycken adresser, där den första används för att skicka meddelanden till styrkretsen, och den andra hanterar meddelanden som skickas ut. Genom att skicka meddelande via CAN-bussen kan styrkretsen läsa in meddelandet och generera de test som skall utföras. Vad som skall utföras tolkas genom att läsa av hur de databytes som skickas via meddelandet är ställda, där varje byte innehåller 8 bitar och vardera bit motsvarar ett relä. En ettställd bit innebär att reläet skall aktiveras, och en nollställd bit innebär avaktivering. I steg 1 kan styrkretsen styra två separata matningsspänningar från varsitt nätaggregat och sluta respektive bryta matningen från aggregaten ut till kretskortet som skall testas. I steg 2 kan styrkretsen styra 14 stycken reläer, som via en D-subkontakt är inkopplade till varsitt pin-par som skall kunna kortslutas, beroende på vilka komponenter på kretskortet som önskas kortslutas. Detta kan modifieras efter önskemål, både hur många reläer som krävs (hur många pin-par som skall gå att kortsluta) samt vad för pin-par det gäller. Detta då modulen inte är anpassad för några särskilda pin-par utan det är kopplingen in till motsvarande D-subkontakt ifrån kretskortet som avgör. Efter att styrkretsen läst in meddelandet och utfört önskad aktivering respektive avaktivering av relän skickas ett statusmeddelande tillbaka. Det görs genom att programmet läser av den pin på styrkretsen som är kopplad till respektive relä, där pinnen antingen kan vara ettställd eller nollställd. Värdet på pinnen läggs sedan in på den platsen i byten som det aktuella reläet påverkades av från början. På så sätt går det att läsa av meddelandet som skickas tillbaka och se vilka relän som är aktiva respektive icke aktiva.

9 Diskussion

Då detta projekt inleddes var ett av målen att upplägget av arbetet skulle efterlikna den arbetsmetodik som tillämpas av ingenjörer för att få en god grund inför det framtida yrkeslivet. Under inledningsfasen av detta projekt utfördes en tidsplanering över vilka moment som skulle utföras samt under vilken tidsperiod dessa var beräknade att genomföras. Tidsplanen, som var i form av ett Gantt-schema, har hela tiden använts för att stämma av hur gruppen förhållit sig tidsmässigt. Tidsplaneringen har upplevts väl utvecklad och därmed enkel att följa.

Tyvärr utfördes inte steg 3 i detta projekt, men tre stycken teoretiska tillvägagångssätt togs dock fram, med anledningen att detta steg är mycket intressant för företaget och att det därmed ges möjlighet att i framtiden vidareutveckla delmomentet. De tre förslagsalternativen är relativt tunt förklarade i rapporten men lämnar möjlighet för vidareutveckling.

Arbetet efterlämnar stora utvecklingsmöjligheter där inte bara steg 3 kan genomföras utan steg 2 även kan byggas på beroende på om kortslutning av ytterligare pin-par önskas. I detta projekt används en 37-pols D-subkontakt, vilket möjliggör kortslutning av 18 stycken olika pin-par. Detta går dock att utöka genom implementering av en kontakt med fler poler alternativt fler kontakter. Det finns även utrymme för modifiering av modulen om annan funktionalitet önskas läggas till. För senare användning är det rekommenderat att den eller de som eventuellt kan komma att vidareutveckla projektet bör bygga in den fullständiga modulen i en sluten låda, för att erhålla en mer hållbar slutprodukt. Det är då lämpligt att i anslutning till kontakten ut till kretskortet koppla en lysdiod som hör ihop med respektive nätaggregat, för att få visuell återkoppling om när spänningen är påkopplad eller inte.

Under projektvecka 9 uttryckte Consat önskemål om att programkoden skulle utföras på ett annorlunda sätt. Detta då de insett att det nya sättet skulle fungera bättre långsiktigt än de tidigare direktiven. Det gjorde de sista veckorna något mer hektiska, men i slutändan är gruppen nöjd med hur programmet såg ut efter ändringen. Dock finns det säkerligen utvecklingspotential för den slutgiltiga koden, med tanke på att tiden blev knapp.

Utöver det som nämns ovan anser vi att projektet i sin helhet har flutit på bra, mycket tack vare väldisponerad tidsplanering, kontinuerlig dokumentation samt bra handledning från både Consat och Chalmers. Något som kunde ha planerats bättre var hårdvarubeställningarna, som stannade upp processen litet med anledning av väntan på leverans. Dock gav detta tillfälle att reflektera över genomförda moment samt se över kommande projektuppgifter, vilket i slutändan kan ses som positivt.

Referenser

Anderson R and Cervo D: *Pro Arduino*, Apress 2013

Arduino: *Arduino™ Mega 2560*. Hämtat från http://www.arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf (Acc 2015-05-05)

Arduino: *Arduino™ Uno Reference Design*. Hämtat från http://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf (Acc 2015-05-05)

Arduino. *Products: Arduino Mega.* Hämtat från http://arduino.cc/en/Main/arduinoBoardMega (Acc 2015-04-02)

Barrett, S and Kridner, J: *Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black (2nd Edition)*, Morgan & Claypool 2013

Beagleboard. *Boards.* Hämtat från http://beagleboard.org/boards (Acc 2015-04-17)

Bilting, U & Jan Skansholm: *Vägen till C (upplaga 4:1)*, Lund, Sverige 2011

Blum, J: *Exploring Arduino: Tools and techniques for engineering wizardry*, Indianapolis, Indiana 2013

Boxall, J: *Arduino workshop – a hands-on introduction with 65 projects*, San Francisco, California 2013

Britannica Academic. *Topic: Voltage Regulator.* Hämtat från http://academic.eb.com.proxy.lib.chalmers.se/EBchecked/topic/632467/voltage-regulator (Acc 2015-04-22)

Broadcom. *Products: BCM2835.* Hämtat från https://www.broadcom.com/products/BCM2835 (Acc 2015-04-16)

Brunnsåker, Peter. Intervju med teknisk handledare Consat Engineering AB. Göteborg 2015

Busmaster. *User Manual (PDF-fil).* Hämtat från http://rbei-etas.github.io/busmaster/ (Acc 2015-05-08)

Bylund, A: *Examensarbete: Teknologier för fordonsdiagnostik*,Institutionen för informationsteknologi, Uppsala Universitet 2009

Cook, M & Andrew Robinson: *Raspberry Pi projects*, Chichester, West Sussex 2014

Cooking-Hacks. *Documentation: Tutorials.* Hämtat från http://www.cooking-hacks.com/documentation/tutorials/raspberry-pi-to-arduino-shields-connection-bridge (Acc 2015-04-16)

Digikey. *Product Search.* Hämtat från http://www.digikey.se/product-search/en?x=0&y=0&lang=en&site=se&KeyWords=102-3262-ND (Acc 2015-04-22)

Electro:Kit. *Hylslist*. Hämtat från http://www.electrokit.com/hylslist-2-54mm-1x8p-stackningsbar.46829 (Acc 2015-04-23)

Electronics Tutorials. *Opamp.* Hämtat från http://www.electronics-tutorials.ws/opamp/opamp_1.html (Acc 2015-05-18)

Github. *Canbus library, SK Pang Electronics, v 1.0 28-03-10*. Hämtat från https://github.com/warewolf/Canbus (Acc 2015-05-05)

Gurevich, V: *Electric relays, principles and applications,* Boca Raton, Florida 2006

Halfacree, G & Eben Upton: *Raspberry Pi user guide (2nd Edition),* Chichester, West Sussex 2013

Lanni, Thomas W: *Programmable Power Supply*. Patent number: 7460381, Laguna Niguel, CA 2008

Lawrence, W: *CAN System Engineering: From Theory to Practical Applications,* London: Springer-Verlag 2013

Microchip Technology Inc: *PIC18(L)F2X/4XK22 Data sheet,* © 2010-2012 Microchip Technology Inc, USA

Onsemi. *Everything you wanted to know about Digital Potentiometer (POT)*. Hämtad från http://www.onsemi.com/pub_link/Collateral/AND8414-D.PDF (Acc 2015-05-18)

Osbeck, M: *Styr och övervakningssystem: Kompendium till projektkurserna LEU743 (Styr- och reglersystem) samt LEU340 (Industriella styr- och övervakningssystem),* Version 2014:nov

Raspberry Pi. *Documentation: Hardware.* Hämtat från https://www.raspberrypi.org/documentation/hardware/raspberrypi/ (Acc 2015-04-16)

Spark docs. *Shields and kits.* Hämtat från http://docs.spark.io/shields/ (Acc 2015-04-09)

Sparkfun: *Canbus_shield-v12, Rev: C*. Hämtat från https://www.sparkfun.com/datasheets/DevTools/Arduino/canbus_shield-v12.pdf (Acc 2015-05-05)

Sparkfun. *Tutorials: How to use a breadboard.* Hämtat från https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard (Acc 2015-04-09)

Sparkfun. *CAN-BUS shield*. Hämtat från https://www.sparkfun.com/products/10039 (Acc 2015-04-20)

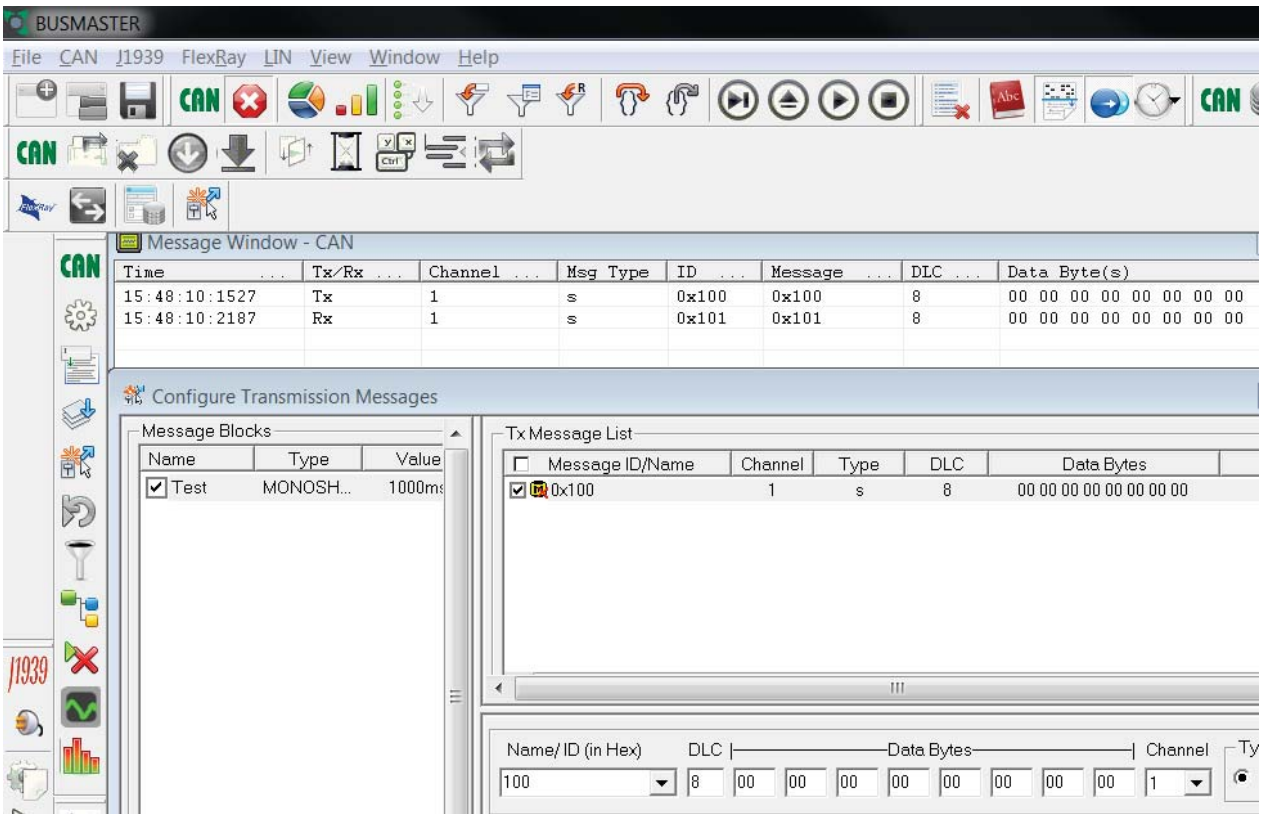
Teufel, B: *Organization of programming languages,* Wien 1991

Wheat, D: *Arduino Internals,* Apress, 2011

Wikipedia. *D-subminiature*. Hämtat från http://en.wikipedia.org/wiki/D-subminiature (Acc 2015-04-23)

Bilagor

Bilaga 1 – Busmaster



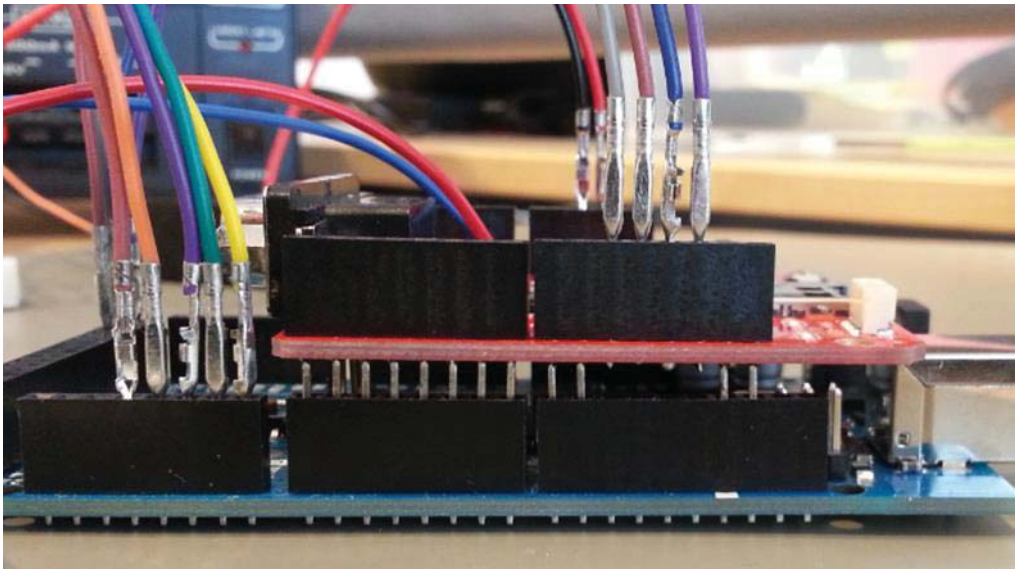
Bilaga 2 – Deklaration över vilken databit som är kopplad till vilket relä

Hårdvara	Benämning programkod	Påverkad byte	Påverkad bit (HEX)	Påverkad bit (BIN)
Shield 1 Relä 1	RELAY1	0	80	1000 0000
Shield 1 Relä 2	RELAY2	0	40	0100 0000
Shield 1 Relä 3	RELAY3	1	80	1000 0000
Shield 1 Relä 4	RELAY4	1	40	0100 0000
Shield 1 Relä 5	RELAY5	1	20	0010 0000
Shield 1 Relä 6	RELAY6	1	10	0001 0000
Shield 1 Relä 7	RELAY7	1	08	0000 1000
Shield 1 Relä 8	RELAY8	1	04	0000 0100
Shield 2 Relä 1	RELAY9	1	02	0000 0010
Shield 2 Relä 2	RELAY10	1	01	0000 0001
Shield 2 Relä 3	RELAY11	2	80	1000 0000
Shield 2 Relä 4	RELAY12	2	40	0100 0000
Shield 2 Relä 5	RELAY13	2	20	0010 0000
Shield 2 Relä 6	RELAY14	2	10	0001 0000
Shield 2 Relä 7	RELAY15	2	08	0000 1000
Shield 2 Relä 8	RELAY16	2	04	0000 0100

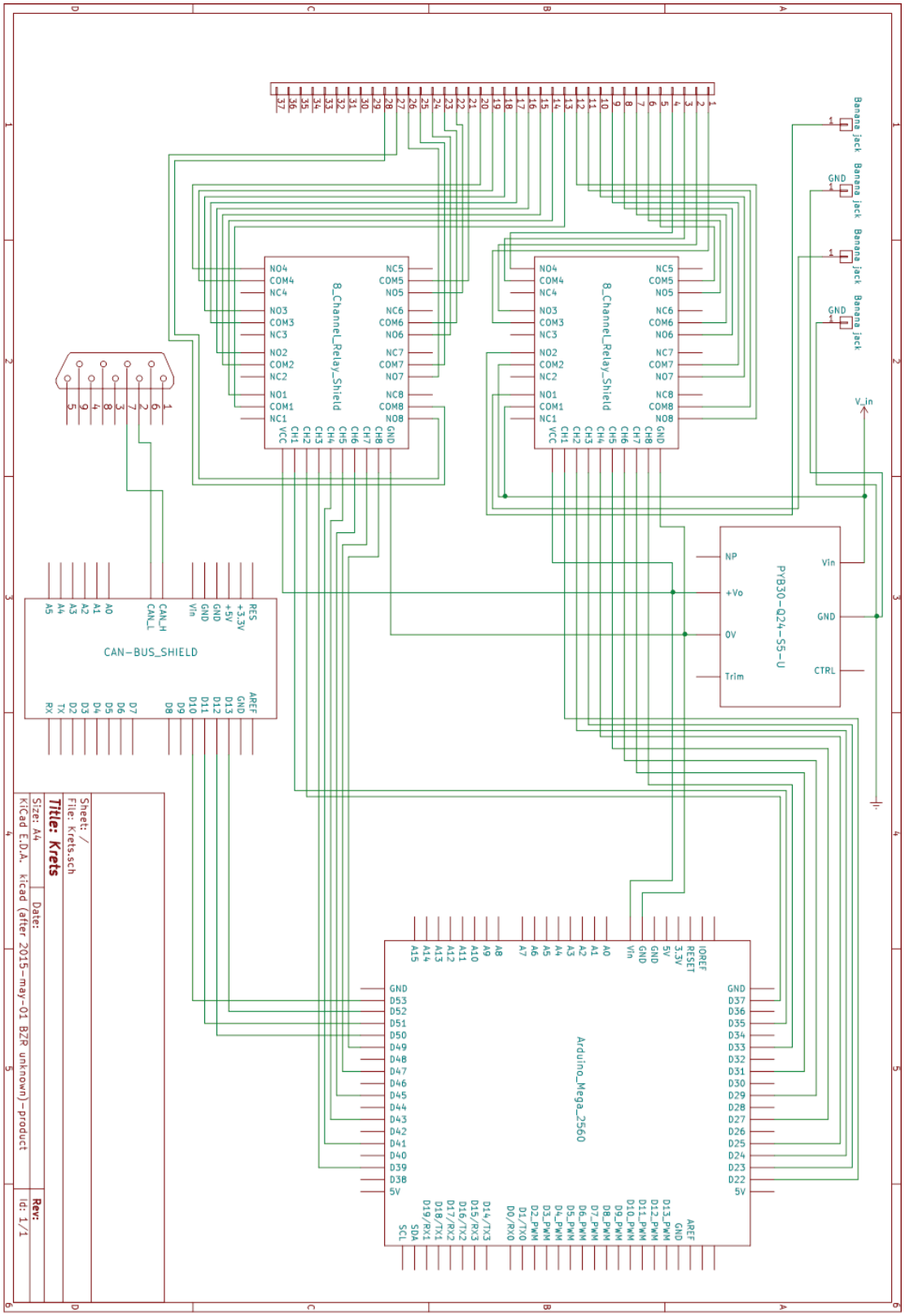
Bilaga 3 – Deklaration över vilka pins på D-subkontakten som kortsluts i Steg 2

Hårdvara	Kopplade pins Dsub
Shield 1 Relä 3	pin 1 & pin 2
Shield 1 Relä 4	pin 3 & pin 4
Shield 1 Relä 5	pin 5 & pin 6
Shield 1 Relä 6	pin 7 & pin 8
Shield 1 Relä 7	pin 9 & pin 10
Shield 1 Relä 8	pin 11 & pin 12
Shield 2 Relä 1	pin 13 & pin 14
Shield 2 Relä 2	pin 15 & pin 16
Shield 2 Relä 3	pin 17 & pin 18
Shield 2 Relä 4	pin 19 & pin 20
Shield 2 Relä 5	pin 21 & pin 22
Shield 2 Relä 6	pin 23 & pin 24
Shield 2 Relä 7	pin 25 & pin 26
Shield 2 Relä8	pin 27 & pin 28

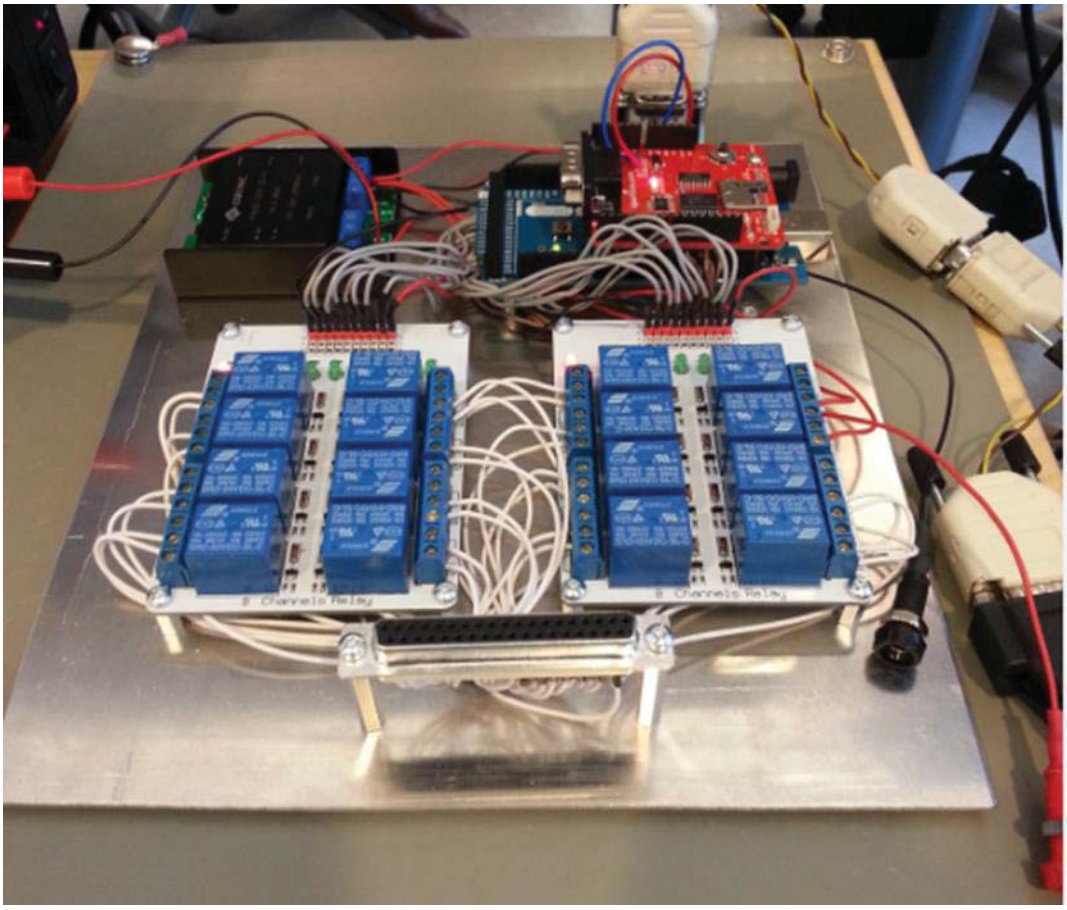
Bilaga 4 – Fotografi på lösning för koppling mellan CAN-shield och Arduino Mega 2560
Bilden visar hur pin 10, 11, 12 och 13 har kapats på CAN-shielden för att undvika kontakt med pin 10, 11, 12 och 13 på Arduinon. Istället görs kopplingen från pin 50, 51, 52 och 53 från Arduinon.
(Författarnas egen bild)



Bilaga 5 – Kretsschema



Bilaga 6 – Fotografi på uppkoppling vid projektets slut



Bilaga 7 - Programkod defaults.h

```
/**
    File: defaults.h
**/

#ifndef DEFAULTS_H
#define DEFAULTS_H

// Arduino Mega 2560 pin definition for SPI-bus
#define P_MOSI          B,2 // pin 51
#define P_MISO          B,3 // pin 50
#define P_SCK           B,1 // pin 52
#define MCP2515_CS      B,0 // pin 53
#define MCP2515_INT     E,4 // pin 2
#define LED2_HIGH       H,5 // pin 8
#define LED2_LOW        H,5 // pin 8

#endif  // DEFAULTS_H
```

Bilaga 8 - Programkod global.h

```
/**
    File: global.h
**/

#ifndef GLOBAL_H
#define GLOBAL_H

// ----------------------------------------------------------------------
```

```
#define true    1
#define false   0

#define True    1
#define False   0

//typedef    _Bool bool;
//typedef   boolean Bool;

// ----------------------------------------------------------------------

#define RESET(x)         _XRS(x)
#define SET(x)           _XS(x)
#define TOGGLE(x)        _XT(x)
#define SET_OUTPUT(x)    _XSO(x)
#define SET_INPUT(x)     _XSI(x)
#define IS_SET(x)        _XR(x)

#define PORT(x)          _port2(x)
#define DDR(x)           _ddr2(x)
#define PIN(x)           _pin2(x)

#define _XRS(x,y)    PORT(x) &= ~(1<<y)
#define _XS(x,y)     PORT(x) |= (1<<y)
#define _XT(x,y)     PORT(x) ^= (1<<y)

#define _XSO(x,y)    DDR(x) |= (1<<y)
#define _XSI(x,y)    DDR(x) &= ~(1<<y)

#define _XR(x,y)     ((PIN(x) & (1<<y)) != 0)

#define _port2(x)    PORT ## x
#define _ddr2(x)     DDR ## x
#define _pin2(x)     PIN ## x


#endif  // GLOBAL_H
```

Bilaga 9 - Programkod mcp2515.c

```
/**
    File: mcp2515.c
**/
/*
 * Copyright (c) 2007 Fabian Greif
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 */
// ----------------------------------------------------------------------

#include <avr/io.h>
#include <util/delay.h>
#include "Arduino.h"
#include "global.h"
#include "mcp2515.h"
#include "mcp2515_defs.h"
#include "defaults.h"

// ----------------------------------------------------------------------
```

```
// Writes / reads a byte from the hardware SPI Bus

uint8_t spi_putc( uint8_t data )
{
    // put byte in send-buffer
    SPDR = data;

    // wait until byte was send
    while( !( SPSR & (1<<SPIF) ) )
        ;

    return SPDR;
}

// ----------------------------------------------------------------------
void mcp2515_write_register( uint8_t adress, uint8_t data )
{
    RESET(MCP2515_CS);

    spi_putc(SPI_WRITE);
    spi_putc(adress);
    spi_putc(data);

    SET(MCP2515_CS);
}

// ----------------------------------------------------------------------
uint8_t mcp2515_read_register(uint8_t adress)
{
    uint8_t data;

    RESET(MCP2515_CS);

    spi_putc(SPI_READ);
    spi_putc(adress);

    data = spi_putc(0xff);

    SET(MCP2515_CS);

    return data;
}

// ----------------------------------------------------------------------
void mcp2515_bit_modify(uint8_t adress, uint8_t mask, uint8_t data)
{
    RESET(MCP2515_CS);

    spi_putc(SPI_BIT_MODIFY);
    spi_putc(adress);
    spi_putc(mask);
    spi_putc(data);

    SET(MCP2515_CS);
}

// ----------------------------------------------------------------------
uint8_t mcp2515_read_status(uint8_t type)
{
    uint8_t data;

    RESET(MCP2515_CS);

    spi_putc(type);
    data = spi_putc(0xff);

    SET(MCP2515_CS);
```

```
    return data;
}

// ----------------------------------------------------------------------
uint8_t mcp2515_init(uint8_t speed)
{
    SET(MCP2515_CS);
    SET_OUTPUT(MCP2515_CS);

    RESET(P_SCK);
    RESET(P_MOSI);
    RESET(P_MISO);

    SET_OUTPUT(P_SCK);
    SET_OUTPUT(P_MOSI);
    SET_INPUT(P_MISO);

    SET_INPUT(MCP2515_INT);
    SET(MCP2515_INT);

    // active SPI master interface
    SPCR = (1<<SPE)|(1<<MSTR) | (0<<SPR1)|(1<<SPR0);
    SPSR = 0;

    // reset MCP2515 by software reset.
    // After this he is in configuration mode.
    RESET(MCP2515_CS);
    spi_putc(SPI_RESET);
    SET(MCP2515_CS);

    // wait a little bit until the MCP2515 has restarted
    _delay_us(10);

    // load CNF1..3 Register
    RESET(MCP2515_CS);
    spi_putc(SPI_WRITE);
    spi_putc(CNF3);

/*  spi_putc((1<<PHSEG21));     // Bitrate 125 kbps at 16 MHz
    spi_putc((1<<BTLMODE)|(1<<PHSEG11));
    spi_putc((1<<BRP2)|(1<<BRP1)|(1<<BRP0));
*/
/*
    spi_putc((1<<PHSEG21));     // Bitrate 250 kbps at 16 MHz
    spi_putc((1<<BTLMODE)|(1<<PHSEG11));
    spi_putc((1<<BRP1)|(1<<BRP0));
*/
    spi_putc((1<<PHSEG21));     // Bitrate 250 kbps at 16 MHz
    spi_putc((1<<BTLMODE)|(1<<PHSEG11));
    //spi_putc(1<<BRP0);
    spi_putc(speed);

    // activate interrupts
    spi_putc((1<<RX1IE)|(1<<RX0IE));
    SET(MCP2515_CS);

    // test if we could read back the value => is the chip accessible?
    if (mcp2515_read_register(CNF1) != speed) {
        SET(LED2_HIGH);

        return false;
    }

    // deactivate the RXnBF Pins (High Impedance State)
    mcp2515_write_register(BFPCTRL, 0);

    // set TXnRTS as inputs
    mcp2515_write_register(TXRTSCTRL, 0);
```

```
    // turn off filters => receive any message
    mcp2515_write_register(RXB0CTRL, (1<<RXM1)|(1<<RXM0));
    mcp2515_write_register(RXB1CTRL, (1<<RXM1)|(1<<RXM0));

    // reset device to normal mode
    mcp2515_write_register(CANCTRL, 0);
//  SET(LED2_HIGH);
    return true;
}

// ----------------------------------------------------------------------
// check if there are any new messages waiting

uint8_t mcp2515_check_message(void) {
    return (!IS_SET(MCP2515_INT));
}

// ----------------------------------------------------------------------
// check if there is a free buffer to send messages

uint8_t mcp2515_check_free_buffer(void)
{
    uint8_t status = mcp2515_read_status(SPI_READ_STATUS);

    if ((status & 0x54) == 0x54) {
        // all buffers used
        return false;
    }

    return true;
}

// ----------------------------------------------------------------------
uint8_t mcp2515_get_message(tCAN *message)
{
    // read status
    uint8_t status = mcp2515_read_status(SPI_RX_STATUS);
    uint8_t addr;
    uint8_t t;
    if (bit_is_set(status,6)) {
        // message in buffer 0
        addr = SPI_READ_RX;
    }
    else if (bit_is_set(status,7)) {
        // message in buffer 1
        addr = SPI_READ_RX | 0x04;
    }
    else {
        // Error: no message available
        return 0;
    }

    RESET(MCP2515_CS);
    spi_putc(addr);

    // read id
    message->id  = (uint16_t) spi_putc(0xff) << 3;
    message->id |=            spi_putc(0xff) >> 5;

    spi_putc(0xff);
    spi_putc(0xff);

    // read DLC (Data Length Code)
    uint8_t length = spi_putc(0xff) & 0x0f;

    message->header.length = length;
    message->header.rtr = (bit_is_set(status, 3)) ? 1 : 0;
```

```
    // read data
    for (t=0;t<length;t++) {
        message->data[t] = spi_putc(0xff);
    }
    SET(MCP2515_CS);

    // clear interrupt flag
    if (bit_is_set(status, 6)) {
        mcp2515_bit_modify(CANINTF, (1<<RX0IF), 0);
    }
    else {
        mcp2515_bit_modify(CANINTF, (1<<RX1IF), 0);
    }

    return (status & 0x07) + 1;
}

// ----------------------------------------------------------------------
uint8_t mcp2515_send_message(tCAN *message)
{
    uint8_t status = mcp2515_read_status(SPI_READ_STATUS);

    /* Statusbyte:
     *
     * Bit  Function
     *  2   TXB0CNTRL.TXREQ
     *  4   TXB1CNTRL.TXREQ
     *  6   TXB2CNTRL.TXREQ
     */
    uint8_t address;
    uint8_t t;
    //SET(LED2_HIGH);
    if (bit_is_clear(status, 2)) {
        address = 0x00;
    }
    else if (bit_is_clear(status, 4)) {
        address = 0x02;
    }
    else if (bit_is_clear(status, 6)) {
        address = 0x04;
    }
    else {
        // all buffer used => could not send message
        return 0;
    }

    RESET(MCP2515_CS);
    spi_putc(SPI_WRITE_TX | address);

    spi_putc(message->id >> 3);
    spi_putc(message->id << 5);

    spi_putc(0);
    spi_putc(0);

    uint8_t length = message->header.length & 0x0f;

    if (message->header.rtr) {
        // a rtr-frame has a length, but contains no data
        spi_putc((1<<RTR) | length);
    }
    else {
        // set message length
        spi_putc(length);

        // data
        for (t=0;t<length;t++) {
```

```
            spi_putc(message->data[t]);
        }
    }
    SET(MCP2515_CS);

    _delay_us(1);

    // send message
    RESET(MCP2515_CS);
    address = (address == 0) ? 1 : address;
    spi_putc(SPI_RTS | address);
    SET(MCP2515_CS);

    return address;
}
```

Bilaga 10 - Programkod mcp2515.h

```
/**
    File: mcp2515.h
**/

#ifndef MCP2515_H
#define MCP2515_H

// ----------------------------------------------------------------------------
/* Copyright (c) 2007 Fabian Greif
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 */
// ----------------------------------------------------------------------------

#include <inttypes.h>

#include "mcp2515_defs.h"
#include "global.h"
#ifdef __cplusplus

extern "C"
{


#endif
// ----------------------------------------------------------------------------
typedef struct
{
    uint16_t id;                //id represents where the data come from
    struct {
        int8_t rtr : 1;         //rtr represents the status of the frame. '0' means
standard frame. '1' means extended frame
        uint8_t length : 4;     //length represents the length of this frame
    } header;
    uint8_t data[8];
} tCAN;

// ----------------------------------------------------------------------------
uint8_t spi_putc( uint8_t data );

// ----------------------------------------------------------------------------
```

```
void mcp2515_write_register( uint8_t adress, uint8_t data );

// ------------------------------------------------------------------------
uint8_t mcp2515_read_register(uint8_t adress);

// ------------------------------------------------------------------------
void mcp2515_bit_modify(uint8_t adress, uint8_t mask, uint8_t data);

// ------------------------------------------------------------------------
uint8_t mcp2515_read_status(uint8_t type);

// ------------------------------------------------------------------------

uint8_t mcp2515_init(uint8_t speed);

// ------------------------------------------------------------------------
// check if there are any new messages waiting
uint8_t mcp2515_check_message(void);

// ------------------------------------------------------------------------
// check if there is a free buffer to send messages
uint8_t mcp2515_check_free_buffer(void);

// ------------------------------------------------------------------------
uint8_t mcp2515_get_message(tCAN *message);

// ------------------------------------------------------------------------
uint8_t mcp2515_send_message(tCAN *message);


#ifdef __cplusplus
}
#endif

#endif  // MCP2515_H
```

Bilaga 11 - Programkod mcp2515_defs.h

```
/**
    File: mcp2515_defs.h
**/

#ifndef MCP2515_DEFS_H
#define MCP2515_DEFS_H

/* Name:    SPI commands */
#define SPI_RESET       0xC0
#define SPI_READ        0x03
#define SPI_READ_RX     0x90
#define SPI_WRITE       0x02
#define SPI_WRITE_TX    0x40
#define SPI_RTS         0x80
#define SPI_READ_STATUS 0xA0
#define SPI_RX_STATUS   0xB0
#define SPI_BIT_MODIFY  0x05

/* Name:    MCP2515 control registers */
#define RXF0SIDH    0x00
#define RXF0SIDL    0x01
#define RXF0EID8    0x02
#define RXF0EID0    0x03
#define RXF1SIDH    0x04
#define RXF1SIDL    0x05
#define RXF1EID8    0x06
#define RXF1EID0    0x07
#define RXF2SIDH    0x08
#define RXF2SIDL    0x09
#define RXF2EID8    0x0A
```

```
#define RXF2EID0    0x0B
#define BFPCTRL     0x0C
#define TXRTSCTRL   0x0D
#define CANSTAT     0x0E
#define CANCTRL     0x0F

#define RXF3SIDH    0x10
#define RXF3SIDL    0x11
#define RXF3EID8    0x12
#define RXF3EID0    0x13
#define RXF4SIDH    0x14
#define RXF4SIDL    0x15
#define RXF4EID8    0x16
#define RXF4EID0    0x17
#define RXF5SIDH    0x18
#define RXF5SIDL    0x19
#define RXF5EID8    0x1A
#define RXF5EID0    0x1B
#define TEC         0x1C
#define REC         0x1D

#define RXM0SIDH    0x20
#define RXM0SIDL    0x21
#define RXM0EID8    0x22
#define RXM0EID0    0x23
#define RXM1SIDH    0x24
#define RXM1SIDL    0x25
#define RXM1EID8    0x26
#define RXM1EID0    0x27
#define CNF3        0x28
#define CNF2        0x29
#define CNF1        0x2A
#define CANINTE     0x2B
#define CANINTF     0x2C
#define EFLG        0x2D

#define TXB0CTRL    0x30
#define TXB0SIDH    0x31
#define TXB0SIDL    0x32
#define TXB0EID8    0x33
#define TXB0EID0    0x34
#define TXB0DLC     0x35
#define TXB0D0      0x36
#define TXB0D1      0x37
#define TXB0D2      0x38
#define TXB0D3      0x39
#define TXB0D4      0x3A
#define TXB0D5      0x3B
#define TXB0D6      0x3C
#define TXB0D7      0x3D

#define TXB1CTRL    0x40
#define TXB1SIDH    0x41
#define TXB1SIDL    0x42
#define TXB1EID8    0x43
#define TXB1EID0    0x44
#define TXB1DLC     0x45
#define TXB1D0      0x46
#define TXB1D1      0x47
#define TXB1D2      0x48
#define TXB1D3      0x49
#define TXB1D4      0x4A
#define TXB1D5      0x4B
#define TXB1D6      0x4C
#define TXB1D7      0x4D

#define TXB2CTRL    0x50
#define TXB2SIDH    0x51
```

```
#define TXB2SIDL    0x52
#define TXB2EID8    0x53
#define TXB2EID0    0x54
#define TXB2DLC     0x55
#define TXB2D0      0x56
#define TXB2D1      0x57
#define TXB2D2      0x58
#define TXB2D3      0x59
#define TXB2D4      0x5A
#define TXB2D5      0x5B
#define TXB2D6      0x5C
#define TXB2D7      0x5D

#define RXB0CTRL    0x60
#define RXB0SIDH    0x61
#define RXB0SIDL    0x62
#define RXB0EID8    0x63
#define RXB0EID0    0x64
#define RXB0DLC     0x65
#define RXB0D0      0x66
#define RXB0D1      0x67
#define RXB0D2      0x68
#define RXB0D3      0x69
#define RXB0D4      0x6A
#define RXB0D5      0x6B
#define RXB0D6      0x6C
#define RXB0D7      0x6D

#define RXB1CTRL    0x70
#define RXB1SIDH    0x71
#define RXB1SIDL    0x72
#define RXB1EID8    0x73
#define RXB1EID0    0x74
#define RXB1DLC     0x75
#define RXB1D0      0x76
#define RXB1D1      0x77
#define RXB1D2      0x78
#define RXB1D3      0x79
#define RXB1D4      0x7A
#define RXB1D5      0x7B
#define RXB1D6      0x7C
#define RXB1D7      0x7D

/* Name:    Bit definition of the various registers */

/** \brief  Bit definition of BFPCTRL */
#define B1BFS       5
#define B0BFS       4
#define B1BFE       3
#define B0BFE       2
#define B1BFM       1
#define B0BFM       0

/** \brief  Bit definition of TXRTSCTRL */
#define B2RTS       5
#define B1RTS       4
#define B0RTS       3
#define B2RTSM      2
#define B1RTSM      1
#define B0RTSM      0

/** \brief  Bit definition of CANSTAT */
#define OPMOD2      7
#define OPMOD1      6
#define OPMOD0      5
#define ICOD2       3
#define ICOD1       2
#define ICOD0       1
```

```
/** \brief  Bit definition of CANCTRL */
#define REQOP2      7
#define REQOP1      6
#define REQOP0      5
#define ABAT        4
#define CLKEN       2
#define CLKPRE1     1
#define CLKPRE0     0

/** \brief  Bit definition of CNF3 */
#define WAKFIL      6
#define PHSEG22     2
#define PHSEG21     1
#define PHSEG20     0

/** \brief  Bit definition of CNF2 */
#define BTLMODE     7
#define SAM         6
#define PHSEG12     5
#define PHSEG11     4
#define PHSEG10     3
#define PHSEG2      2
#define PHSEG1      1
#define PHSEG0      0

/** \brief  Bit definition of CNF1 */
#define SJW1        7   //Synchronization Jump Width
#define SJW0        6
#define BRP5        5   //Baud Rate Prescaler
#define BRP4        4
#define BRP3        3
#define BRP2        2
#define BRP1        1
#define BRP0
/** \brief  Bit definition of CANINTE */
#define MERRE       7
#define WAKIE       6
#define ERRIE       5
#define TX2IE       4
#define TX1IE       3
#define TX0IE       2
#define RX1IE       1
#define RX0IE       0

/** \brief  Bit definition of CANINTF */
#define MERRF       7
#define WAKIF       6
#define ERRIF       5
#define TX2IF       4
#define TX1IF       3
#define TX0IF       2
#define RX1IF       1
#define RX0IF       0

/** \brief  Bit definition of EFLG */
#define RX1OVR      7
#define RX0OVR      6
#define TXB0        5
#define TXEP        4
#define RXEP        3
#define TXWAR       2
#define RXWAR       1
#define EWARN       0

/** \brief  Bit definition of TXBnCTRL (n = 0, 1, 2) */
#define ABTF        6
#define MLOA        5
```

```
#define TXERR       4
#define TXREQ       3
#define TXP1        1
#define TXP0        0

/** \brief  Bit definition of RXB0CTRL */
#define RXM1        6
#define RXM0        5
#define RXRTR       3
#define BUKT        2
#define BUKT1       1
#define FILHIT0     0

/** \brief  Bit definition of TXBnSIDL (n = 0, 1) */
#define EXIDE       3

/**
 * \brief   Bit definition of RXB1CTRL
 * \see     RXM1, RXM0, RXRTR and FILHIT0 are already defined for RXB0CTRL
 */
#define FILHIT2     2
#define FILHIT1     1

/** \brief  Bit definition of RXBnSIDL (n = 0, 1) */
#define SRR         4
#define IDE         3

/**
 * \brief   Bit definition of RXBnDLC (n = 0, 1)
 * \see     TXBnDLC   (same bits)
 */
#define RTR         6
#define DLC3        3
#define DLC2        2
#define DLC1        1
#define DLC0        0
#endif  // MCP2515_DEFS_H
```

Bilaga 12 - Programkod Canbus.h

```
/**
File: Canbus.h
**/

#ifndef canbus__h
#define canbus__h

#define RECEIVE_FRAME_ID    0x100   //Identifier for get message
#define SEND_FRAME_ID       0X101   //Identifier for send message

#define CANSPEED_125    7       // CAN speed at 125 kbps
#define CANSPEED_250    3       // CAN speed at 250 kbps
#define CANSPEED_500    1       // CAN speed at 500 kbps

//Definitions for step 1
#define RELAY1 22               //Set RELAY1 to pin 22
#define RELAY2 24               //Set RELAY2 to pin 24

//Definition for step 2
#define RELAY3 23               //Set RELAY3 to pin 23
#define RELAY4 25               //Set RELAY4 to pin 25
#define RELAY5 27               //Set RELAY5 to pin 27
#define RELAY6 29               //Set RELAY6 to pin 29
#define RELAY7 31               //Set RELAY7 to pin 31
#define RELAY8 33               //Set RELAY8 to pin 33
#define RELAY9 35               //Set RELAY9 to pin 35
#define RELAY10 37              //Set RELAY10 to pin 37
#define RELAY11 39              //Set RELAY11 to pin 39
```

```
#define RELAY12 41              //Set RELAY12 to pin 41
#define RELAY13 43              //Set RELAY13 to pin 43
#define RELAY14 45              //Set RELAY14 to pin 45
#define RELAY15 47              //Set RELAY15 to pin 47
#define RELAY16 49              //Set RELAY16 to pin 49

 /*Uncomment to implement more relays
#define RELAY17 26              //Set RELAY17 to pin 26
#define RELAY18 28              //Set RELAY18 to pin 28
#define RELAY19 30              //Set RELAY19 to pin 30
#define RELAY20 32              //Set RELAY20 to pin 32
#define RELAY21 34              //Set RELAY21 to pin 34
#define RELAY22 36              //Set RELAY22 to pin 36
#define RELAY23 38              //Set RELAY23 to pin 38
#define RELAY24 40              //Set RELAY24 to pin 40
#define RELAY25 42              //Set RELAY25 to pin 42
#define RELAY26 44              //Set RELAY26 to pin 44 */

//The class CanbusClass consisting the variables needed, all public
class CanbusClass
{
  public:
    CanbusClass();
    char init(unsigned char);
    char ecu_req(char *buffer);
private:

};
extern CanbusClass Canbus;

#endif
```

Bilaga 13 - Programkod Canbus.cpp

```
/**
File: Canbus.cpp
Includes main programme from line 24
**/

#include "Arduino.h"
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "pins_arduino.h"
#include <inttypes.h>
#include "global.h"
#include "mcp2515.h"
#include "defaults.h"
#include "Canbus.h"


/* C++ wrapper */
CanbusClass::CanbusClass() {

}

char CanbusClass::ecu_req(char *buffer)
{
    tCAN message;

    //Check for received frames
    if (mcp2515_check_message())
    {
        //Get data (&-sign means get the address for 'message')
        if (mcp2515_get_message(&message))
        {
            //Do only if message identifier equals RECEIVE_FRAME_ID
```

```
if(RECEIVE_FRAME_ID == message.id)
{

    //********************STEP 1*******************//
    if(0 != (message.data[0] & 0x80)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY1, HIGH); //Sets power supply 1 to on
        Serial.print("Power supply 1 activated. \n");
    }
    else                            //If bitwise AND resolves to false
    {
    digitalWrite(RELAY1, LOW);      //Sets power supply 1 to off
    Serial.print("Power supply 1 deactivated. \n");
    }

    if(0 != (message.data[0] & 0x40)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY2, HIGH); //Sets power supply 2 to on
        Serial.print("Power supply 2 activated. \n");
    }
    else                            //If bitwise AND resolves to false
    {
        digitalWrite(RELAY2, LOW);  //Sets power supply 2 to off
        Serial.print("Power supply 2 deactivated. \n");
    }

    //Bitwise mask and shift left to put bit in the right position
    message.data[0] |= ((digitalRead(RELAY1) & 0x01) << 7);
    message.data[0] |= ((digitalRead(RELAY2) & 0x01) << 6);


    //********************STEP 2*******************//
    if(0 != (message.data[1] & 0x80)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY3, HIGH); //Creates short between pin 0 & 1
        Serial.print("Short circuit pin 0 and 1. \n");
    }
    else                            //If bitwise AND resolves to false
    {
        digitalWrite(RELAY3, LOW);  //Deactivate short
        Serial.print("Short circuit pin 0 and 1 off. \n");
    }

    if(0 != (message.data[1] & 0x40)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY4, HIGH); //Creates short between pin 2 & 3
        Serial.print("Short circuit pin 2 and 3. \n");
    }
    else                            //If bitwise AND resolves to false
    {
        digitalWrite(RELAY4, LOW);  //Deactivate short
        Serial.print("Short circuit pin 2 and 3 off. \n");
    }

    if(0 != (message.data[1] & 0x20)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY5, HIGH); //Creates short between pin 4 & 5
        Serial.print("Short circuit pin 4 and 5. \n");
    }
    else                            //If bitwise AND resolves to false
    {
        digitalWrite(RELAY5, LOW);  //Deactivate short
        Serial.print("Short circuit pin 4 and 5 off. \n");
    }

    if(0 != (message.data[1] & 0x10)) //If bitwise AND resolves to true
    {
        digitalWrite(RELAY6, HIGH); //Creates short between pin 6 & 7
```

```
    Serial.print("Short circuit pin 6 and 7. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY6, LOW);  //Deactivate short
    Serial.print("Short circuit pin 6 and 7 off. \n");
}

if(0 != (message.data[1] & 0x08)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY7, HIGH); //Creates short between pin 8 & 9
    Serial.print("Short circuit pin 8 and 9. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY7, LOW);  //Deactivate short
    Serial.print("Short circuit pin 8 and 9 off. \n");
}

if(0 != (message.data[1] & 0x04)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY8, HIGH); //Creates short between pin 10 & 11
    Serial.print("Short circuit pin 10 and 11. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY8, LOW);  //Deactivate short
    Serial.print("Short circuit pin 10 and 11 off. \n");
}

//Bitwise mask and shift left to put bit in the right position
message.data[1] |= ((digitalRead(RELAY3) & 0x01) << 7);
message.data[1] |= ((digitalRead(RELAY4) & 0x01) << 6);
message.data[1] |= ((digitalRead(RELAY5) & 0x01) << 5);
message.data[1] |= ((digitalRead(RELAY6) & 0x01) << 4);
message.data[1] |= ((digitalRead(RELAY7) & 0x01) << 3);
message.data[1] |= ((digitalRead(RELAY8) & 0x01) << 2);


if(0 != (message.data[1] & 0x02)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY9, HIGH); //Creates short between pin 12 & 13
    Serial.print("Short circuit pin 12 and 13. \n");
}
else                             //If bitwise AND resolves to false
{
    digitalWrite(RELAY9, LOW);  //Deactivate short
    Serial.print("Short circuit pin 12 and 13 off. \n");
}

if(0 != (message.data[1] & 0x01)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY10, HIGH);//Creates short between pin 14 & 15
    Serial.print("Short circuit pin 14 and 15. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY10, LOW); //Deactivate short
    Serial.print("Short circuit pin 14 and 15 off. \n");
}

//Bitwise mask and shift left to put bit in the right position
message.data[1] |= ((digitalRead(RELAY9) & 0x01) << 1);     //
message.data[1] |= (digitalRead(RELAY10) & 0x01);           //


if(0 != (message.data[2] & 0x80)) //If bitwise AND resolves to true
{
```

```
    digitalWrite(RELAY11, HIGH);//Creates short between pin 16 & 17
    Serial.print("Short circuit pin 16 and 17. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY11, LOW);  //Deactivate short
    Serial.print("Short circuit pin 16 and 17 off. \n");
}

if(0 != (message.data[2] & 0x40)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY12, HIGH);//Creates short between pin 18 & 19
    Serial.print("Short circuit pin 18 and 19. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY12, LOW);  //Deactivate short
    Serial.print("Short circuit pin 18 and 19 off. \n");
}

if(0 != (message.data[2] & 0x20)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY13, HIGH);//Creates short between pin 20 & 21
    Serial.print("Short circuit pin 20 and 21. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY13, LOW); //Deactivate short
    Serial.print("Short circuit pin 20 and 21 off. \n");
}

if(0 != (message.data[2] & 0x10)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY14, HIGH);//Creates short between pin 22 & 23
    Serial.print("Short circuit pin 22 and 23. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY14, LOW); //Deactivate short
    Serial.print("Short circuit pin 22 and 23 off. \n");
}

if(0 != (message.data[2] & 0x08)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY15, HIGH);//Creates short between pin 24 & 25
    Serial.print("Short circuit pin 24 and 25. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY15, LOW);  //Deactivate short
    Serial.print("Short circuit pin 24 and 25 off. \n");
}

if(0 != (message.data[2] & 0x04)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY16, HIGH);//Creates short between pin 26 & 27
    Serial.print("Short circuit pin 26 and 27. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY16, LOW); //Deactivate short
    Serial.print("Short circuit pin 26 and 27 off. \n");
}

//Bitwise mask and shift left to put bit in the right position
message.data[2] |= ((digitalRead(RELAY11) & 0x01) << 7);
message.data[2] |= ((digitalRead(RELAY12) & 0x01) << 6);
message.data[2] |= ((digitalRead(RELAY13) & 0x01) << 5);
```

```
message.data[2] |= ((digitalRead(RELAY14) & 0x01) << 4);
message.data[2] |= ((digitalRead(RELAY15) & 0x01) << 3);
message.data[2] |= ((digitalRead(RELAY16) & 0x01) << 2);
//********************END OF IMPLEMENTED RELAYS******************


/*Uncomment to implement more relays
if(0 != (message.data[2] & 0x02)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY17, HIGH);//Creates short between pin 28 & 29
    Serial.print("Short circuit pin 28 and 29. \n");
}
else                              //If bitwise AND resolves to false
{
    digitalWrite(RELAY17, LOW); //Deactivate short
    Serial.print("Short circuit pin 28 and 29 off. \n");
}

if(0 != (message.data[2] & 0x01)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY18, HIGH);//Creates short between pin 30 & 31
    Serial.print("Short circuit pin 30 and 31. \n");
}
else                              //If bitwise AND resolves to false
{
    digitalWrite(RELAY18, LOW);  //Deactivate short
    Serial.print("Short circuit pin 30 and 31 off. \n");
}

//Bitwise mask and shift left to put bit in the right position
message.data[2] |= ((digitalRead(RELAY17) & 0x01) << 1);
message.data[2] |= (digitalRead(RELAY18) & 0x01);

if(0 != (message.data[3] & 0x80)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY19, HIGH);//Creates short between pin 32 & 33
    Serial.print("Short circuit pin 32 and 33. \n");
}
else                              //If bitwise AND resolves to false
{
    digitalWrite(RELAY19, LOW);  //Deactivate short
    Serial.print("Short circuit pin 32 and 33 off. \n");
}

if(0 != (message.data[3] & 0x40)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY20, HIGH);//Creates short between pin 34 & 35
    Serial.print("Short circuit pin 34 and 35. \n");
}
else                              //If bitwise AND resolves to false
{
    digitalWrite(RELAY20, LOW);  //Deactivate short
    Serial.print("Short circuit pin 34 and 35 off. \n");
}

if(0 != (message.data[3] & 0x20)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY21, HIGH);//Creates short between pin 36 & 37
    Serial.print("Short circuit pin 36 and 37. \n");
}
else                              //If bitwise AND resolves to false
{
    digitalWrite(RELAY21, LOW); //Deactivate short
    Serial.print("Short circuit pin 36 and 37 off. \n");
}

if(0 != (message.data[3] & 0x10)) //If bitwise AND resolves to true
{
```

```
    digitalWrite(RELAY22, HIGH);//Creates short between pin 38 & 39
    Serial.print("Short circuit pin 38 and 39. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY22, LOW); //Deactivate short
    Serial.print("Short circuit pin 38 and 39 off. \n");
}

if(0 != (message.data[3] & 0x08)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY23, HIGH);//Creates short between pin 40 & 41
    Serial.print("Short circuit pin 40 and 41. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY23, LOW); //Deactivate short
    Serial.print("Short circuit pin 40 and 41 off. \n");
}

if(0 != (message.data[3] & 0x04)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY24, HIGH);//Creates short between pin 42 & 43
    Serial.print("Short circuit pin 42 and 43. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY24, LOW);  //Deactivate short
    Serial.print("Short circuit pin 42 and 43 off. \n");
}

if(0 != (message.data[3] & 0x02)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY25, HIGH);//Creates short between pin 44 & 45
    Serial.print("Short circuit pin 44 and 45. \n");
}
else                            //If bitwise AND resolves to false
{
    digitalWrite(RELAY25, LOW);  //Deactivate short
    Serial.print("Short circuit pin 44 and 45 off. \n");
}

if(0 != (message.data[3] & 0x01)) //If bitwise AND resolves to true
{
    digitalWrite(RELAY26, HIGH);//Creates short between pin 46 & 47
    Serial.print("Short circuit pin 46 and 47. \n");
}
else    //If bitwise AND resolves to false
{
    digitalWrite(RELAY26, LOW);  //Deactivate short
    Serial.print("Short circuit pin 46 and 47 off. \n");
}

//Bitwise mask and shift left to put bit in the right position
message.data[3] |= ((digitalRead(RELAY19) & 0x01) << 7);
message.data[3] |= ((digitalRead(RELAY20) & 0x01) << 6);
message.data[3] |= ((digitalRead(RELAY21) & 0x01) << 5);
message.data[3] |= ((digitalRead(RELAY22) & 0x01) << 4);
message.data[3] |= ((digitalRead(RELAY23) & 0x01) << 3);
message.data[3] |= ((digitalRead(RELAY24) & 0x01) << 2);
message.data[3] |= ((digitalRead(RELAY25) & 0x01) << 1);
message.data[3] |= (digitalRead(RELAY26) & 0x01);  */


//Set message identifier to SEND_FRAME_ID
message.id = SEND_FRAME_ID;
//Returns message with information about pin status
mcp2515_send_message(&message);                }
```

```
        }
    }
}

char CanbusClass::init(unsigned char speed)
{
    return mcp2515_init(speed);
}

CanbusClass Canbus;
```

Bilaga 14 - Programkod ecu_reader.pde

```
/**
    File: ecu_reader.pde
    This shield contains the MCP2515 CAN controller and the MCP2551 CAN-bus driver.
**/

#include <Canbus.h>

int LED2 = 8;
int LED3 = 7;

char buffer[16];

void setup() {

  pinMode(LED2, OUTPUT);         //Set pin 8 to output
  pinMode(LED3, OUTPUT);         //Set pin 7 to output
  pinMode(RELAY1, OUTPUT);       //Set pin 22 to output
  pinMode(RELAY2, OUTPUT);       //Set pin 24 to output
  pinMode(RELAY3, OUTPUT);       //Set pin 23 to output
  pinMode(RELAY4, OUTPUT);       //Set pin 25 to output
  pinMode(RELAY5, OUTPUT);       //Set pin 27 to output
  pinMode(RELAY6, OUTPUT);       //Set pin 29 to output
  pinMode(RELAY7, OUTPUT);       //Set pin 31 to output
  pinMode(RELAY8, OUTPUT);       //Set pin 33 to output
  pinMode(RELAY9, OUTPUT);       //Set pin 35 to output
  pinMode(RELAY10, OUTPUT);          //Set pin 37 to output
  pinMode(RELAY11, OUTPUT);          //Set pin 39 to output
  pinMode(RELAY12, OUTPUT);          //Set pin 41 to output
  pinMode(RELAY13, OUTPUT);          //Set pin 43 to output
  pinMode(RELAY14, OUTPUT);          //Set pin 45 to output
  pinMode(RELAY15, OUTPUT);          //Set pin 47 to output
  pinMode(RELAY16, OUTPUT);          //Set pin 49 to output

  /* Uncomment to implement more relays
  pinMode(RELAY17, OUTPUT);          //Set pin 26 to output
  pinMode(RELAY18, OUTPUT);          //Set pin 28 to output
  pinMode(RELAY19, OUTPUT);          //Set pin 30 to output
  pinMode(RELAY20, OUTPUT);          //Set pin 32 to output
  pinMode(RELAY21, OUTPUT);          //Set pin 34 to output
  pinMode(RELAY22, OUTPUT);          //Set pin 36 to output
  pinMode(RELAY23, OUTPUT);          //Set pin 38 to output
  pinMode(RELAY24, OUTPUT);          //Set pin 40 to output
  pinMode(RELAY25, OUTPUT);          //Set pin 42 to output
  pinMode(RELAY26, OUTPUT);          //Set pin 44 to output */

  Serial.begin(115200);              //Sets the data rate in bits per second (baud)
for serial data transmission.
  Serial.println("ECU Reader");      //For debug use

  if(Canbus.init(CANSPEED_500))      //Initialise MCP2515 CAN controller at the
specified speed
  {
    Serial.print("CAN Init OK! \n");
  }
  else
```

```
  {
    Serial.print("Can't init CAN.");
  }

  delay(1000);
}

void loop() {

  Canbus.ecu_req(buffer);           //Request for main programme
  digitalWrite(LED3, HIGH);         //LED3 placed on CAN shield indicates that
programme is running
  delay(250);
  digitalWrite(LED3, LOW);
  delay(250);

}
```

Bilaga 15 – Programkod version 1

Bilaga 9 innehåller den programkod som gruppen inledningsvis skrev. Denna program kod kom sedan att ändras vilken finns redovisad i Bilaga 7.

```
//Definition of addresses
#define CURRENT1_ON         0x05
#define CURRENT1_OFF        0x0C
#define CURRENT2_ON         0x0D
#define CURRENT2_OFF        0x10
#define SHORT_CIRCUIT_1     0x14
#define SHORT_CIRCUIT_2     0x0A
#define SHORT_CIRCUIT_3     0x04
#define SHORT_CIRCUIT_4     0x06
#define SHORT_CIRCUIT_5     0x07
#define SHORT_CIRCUIT_6     0x08
#define CHECK_STATUS_POWER  0x09


//char CanbusClass::ecu_req(char *buffer)
char CanbusClass::ecu_req(unsigned char pid,  char *buffer)
{
    tCAN message;
    float canbus_data;

    // Prepare message
    message.id = PID_REQUEST;
    message.header.rtr = 0;     //status: standard frame
    message.header.length = 8;
    message.data[0] = 0x02;
    message.data[1] = 0x01;
    message.data[2] = pid;
    message.data[3] = 0x00;
    message.data[4] = 0x00;
    message.data[5] = 0x00;
    message.data[6] = 0x00;
    message.data[7] = 0x00;


    if (mcp2515_check_message())
    {
        //&-sign means get the address for 'message'
        if (mcp2515_get_message(&message))
        {
            switch(message.data[2])
                {
                    //********************STEP 1******************
                    case CURRENT1_ON:           //[Power supply 1 on]
                        digitalWrite(RELAY8, HIGH);
                        Serial.print("Power supply 1 activated. \n");
```

```
    message.data[7] = digitalRead(RELAY8);
    mcp2515_send_message(&message);
break;

case CURRENT1_OFF:          //[Power supply 1 off]
    digitalWrite(RELAY8, LOW);
    Serial.print("Power supply 1 deactivated. \n");
    message.data[7] = digitalRead(RELAY8);
    mcp2515_send_message(&message);
break;

case CURRENT2_ON:           //[Power supply 2 on]
    digitalWrite(RELAY7, HIGH);
    Serial.print("Power supply 2 activated. \n");
    message.data[7] = digitalRead(RELAY7);
    mcp2515_send_message(&message);
break;

case CURRENT2_OFF:          //[Power supply 2 off]
    digitalWrite(RELAY7, LOW);
    Serial.print("Power supply 2 deactivated. \n");
    message.data[7] = digitalRead(RELAY7);
    mcp2515_send_message(&message);
break;


//********************STEP 2******************
case SHORT_CIRCUIT_1:                   //[Short circuit 1]
    //Creates short between pin 0 and 1
      digitalWrite(RELAY2, HIGH);
    Serial.print("Short circuit pin 0 and 1. \n");
    message.data[7] = digitalRead(RELAY2);
    mcp2515_send_message(&message);
    delay(2000);                        //Delay 2s
    digitalWrite(RELAY2, LOW);          //Deactivate short
    message.data[7] = digitalRead(RELAY2);
    mcp2515_send_message(&message);
break;

case SHORT_CIRCUIT_2:       //[Short circuit 2]
    //Creates short between pin 2 and 3
      digitalWrite(RELAY3, HIGH);
    Serial.print("Short circuit pin 2 and 3. \n");
    message.data[7] = digitalRead(RELAY3);
    mcp2515_send_message(&message);
    delay(2000);                        //Delay 2s
    digitalWrite(RELAY3, LOW);          //Deactivate short
    message.data[7] = digitalRead(RELAY3);
    mcp2515_send_message(&message);
break;

case SHORT_CIRCUIT_3:                   //[Short circuit 2]
    //Creates short between pin 4 and 5
      digitalWrite(RELAY4, HIGH);
    Serial.print("Short circuit pin 4 and 5. \n");
    message.data[7] = digitalRead(RELAY4);
    mcp2515_send_message(&message);
    delay(2000);                        //Delay 2s
    digitalWrite(RELAY4, LOW);          //Deactivate short
    message.data[7] = digitalRead(RELAY4);
    mcp2515_send_message(&message);
break;

case SHORT_CIRCUIT_4:                   //[Short circuit 2]
    //Creates short between pin 4 and 5
    digitalWrite(RELAY1, HIGH);
    Serial.print("Short circuit pin 6 and 7. \n");
    message.data[7] = digitalRead(RELAY1);
```

```
                    mcp2515_send_message(&message);
                    delay(2000);                        //Delay 2s
                    digitalWrite(RELAY1, LOW);          //Deactivate short
                    message.data[7] = digitalRead(RELAY1);
                    (mcp2515_send_message(&message));
                break;

                case SHORT_CIRCUIT_5:                   //[Short circuit 2]
                    //Creates short between pin 4 and 5
                    digitalWrite(RELAY5, HIGH);
                    Serial.print("Short circuit pin 8 and 9. \n");
                    message.data[7] = digitalRead(RELAY5);
                    mcp2515_send_message(&message);
                    delay(2000);                        //Delay 2s
                    digitalWrite(RELAY5, LOW);          //Deactivate short
                    message.data[7] = digitalRead(RELAY5);
                    mcp2515_send_message(&message);
                break;

                case SHORT_CIRCUIT_6:                   //[Short circuit 2]
                    //Creates short between pin 4 and 5
                    digitalWrite(RELAY6, HIGH);
                    Serial.print("Short circuit pin 10 and 11. \n");
                    message.data[7] = digitalRead(RELAY6);
                    mcp2515_send_message(&message);
                    delay(2000);                        //Delay 2s
                    digitalWrite(RELAY6, LOW);          //Deactivate short
                    message.data[7] = digitalRead(RELAY6);
                    mcp2515_send_message(&message);
                break;


                //********************STATUS******************
                //Check which pins are set to HIGH
                case CHECK_STATUS_POWER:
                        message.data[6] = digitalRead(RELAY8);
                        message.data[7] = digitalRead(RELAY7);
                        if(digitalRead(RELAY8)==1 && digitalRead(RELAY7)==0)
                        {
                            Serial.print("Power supply 1 is active\n");
                        }
                        if(digitalRead(RELAY8)==0 && digitalRead(RELAY7)==1)
                        {
                            Serial.print("Power supply 2 is active\n");
                        }
                        if(digitalRead(RELAY8)==1 && digitalRead(RELAY7)==1)
                        {
                            Serial.print("Both power supplies are active\n");
                        }
                        if(digitalRead(RELAY8)==0 && digitalRead(RELAY7)==0)
                        {
                            Serial.print("No power supply is active.\n");
                        }
                        mcp2515_send_message(&message)

                break;
            }
        }
        else
        {
            Serial.print("Can't read message.\n\n");
        }
    return 1;
    }
}
```