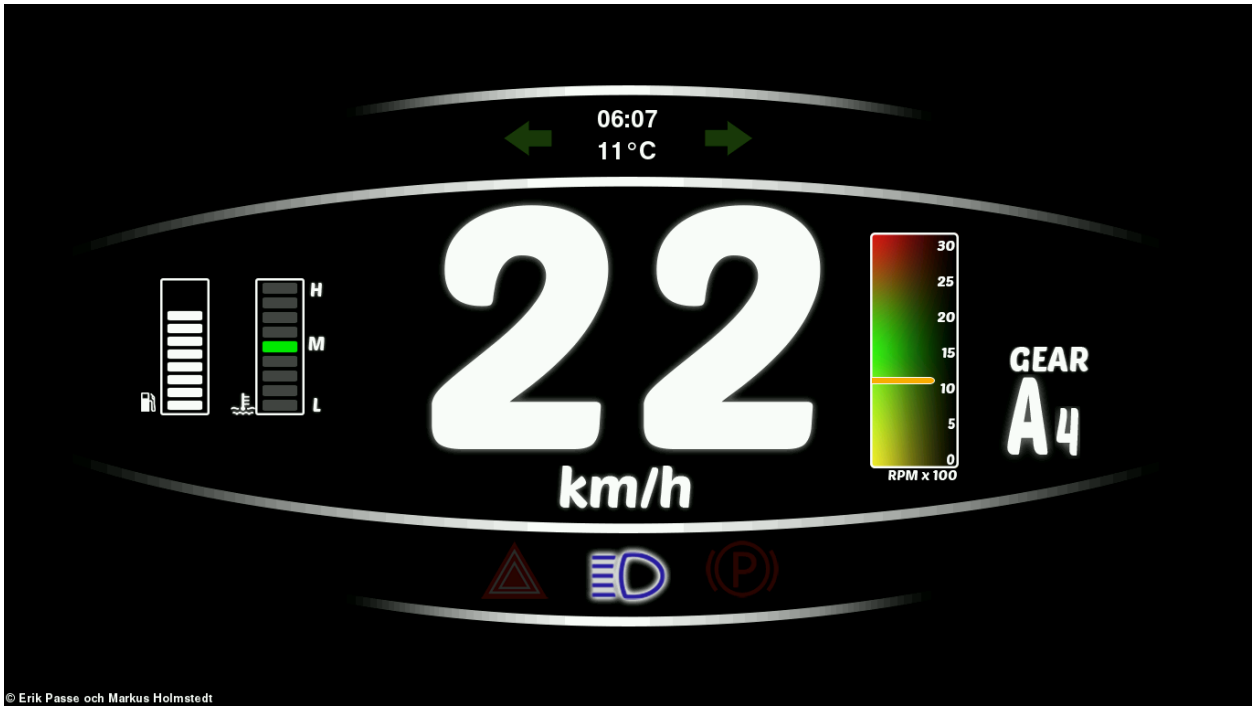
CHALMERS



Prototyp för duplicering av instrumentpanel anpassad för övningskörning av buss

Prototype for duplication of a control panel for a bus in driving practice

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

Erik Påsse
Markus Holmstedt

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

Examensarbete 2015

Prototyp för duplicering av instrumentpanel anpassad för övningskörning av buss

Prototype for duplication of a control panel for a bus in driving practice

Erik Påsse
Markus Holmstedt

CHALMERS

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

Prototyp för duplicering av instrumentpanel anpassad för övningskörning av buss
Prototype for duplication of a control panel for a bus in driving practice
Erik Påsse
Markus Holmstedt

Handledare: Morgan Osbeck, Signaler och System
Examinator: Manne Stenberg, Signaler och System

Examensarbete 2015
Institutionen för signaler och system
CHALMERS TEKNISKA HÖGSKOLA
SE-412 96 Göteborg
Telefonnummer +46 31 772 1000

Omslag: Bild på den framtagna panelen.

Typeset in LaTeX
Göteborg, Sverige 2015

Förord

Denna rapport har skapats under genomförandet av ett examensarbete vid Chalmers tekniska högskola utfört åt Benteler Engineering Services AB med inriktning mot bussar. Arbetet har genomförts av två studenter på Mekatronikprogrammet 180 hp och omfattar 15 hp. Huvuddelen av arbetet har skett på Bentelers kontor på Nya Varvet, Västra Frölunda. Arbetet har inneburit att vi har använt främst programmerings- och elektronikkunskaper vi har införskaffat under de senaste tre åren. Vi har haft väldigt roliga och lärorika veckor.

Vi vill ge ett tack till:

- Morgan Osbeck, handledare på Chalmers
- David Anberg, handledare på Benteler

Trevlig läsning!
Erik Påsse & Markus Holmstedt, Göteborg, Juni 2015

Sammanfattning

När man övningskör i en buss idag så sitter övningsledaren väldigt långt ifrån eleven om man jämför med i en personbil. Det gör att det är svårare att kunna övervaka körningen och förebygga olyckor. Det vore en fördel att kunna ge övningsledaren samma möjligeter som en vanlig förare. Dock används övningskörningsbussar i så liten skala så att det blir ekonomiskt omotiverat att göra en specialbuss som är permanent dubbelstyrd. Därför vill man ta fram en produkt som går att installera och avinstallera i befintliga bussar relativt enkelt. Detta arbete innefattar instrumentpanelens roll i att kunna göra övningsledaren mer reaktiv i övningskörningen. Arbetet har lett fram till en färdig prototyp som väntar på en slutkunds engagemang. Prototypen kan lyssna på bussens interna nätverk och visa den information som finns där, dock inte skriva ut information på nätverket och styra enheter. För att kunna styra enheter på nätverket behövs en slutkund som kan lägga till vår enhet i det befintliga nätverket. Arbetet har gjorts på Benteler Engineering Services kontor i Göteborg. Arbetet har lett fram till en prototyp som är redo funktionellt för att testas i en buss. Dock behövs en anpassning från busstillverkaren för att produkten ska fungera fullt ut.

Nyckelord: Controller Area Network, CAN, CAN-buss, Raspberry Pi, Python, Buss

Prototype for duplication of a control panel for a bus in driving practice
Erik Påsse och Markus Holmstedt
Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Abstract

When you practice drive a bus the instructor sits far from the learner if compared to a car. This makes it harder to oversee the driving and prevent accidents during practice. To make this safe you want to give the instructor the same opportunities as the learner driver. But driving practice for buses is is uncommon and therefore it is economically unjustifiably to make a special bus with permanent dual command. Therefore you want a product that you can use when it is necessary and have the ability to take it away when you need the bus in everyday traffic. This report is about the control panel duplication in the dual command. This work ends in a prototype-state where it needs a bus manufacturer to become involved and accept the product in the ordinary CAN network. The prototype can today read the CAN network and show the information on a display, but not take input or write to the CAN network. All of the work has been done at Benteler Engineering Services office in Gothenburg. At the end of the work the product will be presented as a proof of concept that is ready to be tested in a bus after adjustments made by the manufacturer of the bus.

Keywords: Controller Area Network, CAN, CAN-bus, Raspberry Pi, Python, Bus

Innehåll

Beteckningar

Intercitybuss - Bussar som används på busslinjer som kör i högre hastigheter
RPi - Raspberry Pi
SPI - Serial Peripheral Interface
CAN - Controller Area Network
CAN-buss - Kopplingen mellan enheter i CAN
Nod - Enhet inkopplad på CAN
frame - Alla delar i ett CAN-meddelande
I/O - Input/Output
USB - Universal Serial Bus
C-språk - Programmeringsspråket C
Java - Programmeringsspråket Java
Python - Programmeringsspråket Python
Python-modul - En fil som innehåller pythonkod
HDMI - High-Definition Multimedia Interface
DB9 - Seriell kontakt med 9 pinnar
ODB2 - On-board diagnostics kontakt som används inom fordonsindustrin
GPIO - General-purpose input/output
DLC - Data length code
CANalyser - Ett verktyg för att bland annat spela in och spela upp CAN-meddelanden.

Figurer

Samtliga bilder är skapade av författarna själva.

1
Inledning

1.1 Bakgrund

Benteler Engineering Services AB vill ta fram ett koncept till ett dubbelkommando för vanliga intercitybussar som kan installeras vid behov av övningskörning. Parallellt med detta arbete pågår det ett examensarbete på Benteler inom samma område som behandlar de mekaniska delarna. Vår uppgift är att ta fram ett exempel på hur instrumentpanelen kan utformas, inklusive kommunikation med både körskoleläraren och bussen. Efter projektet vill Benteler kunna visa för Volvo Bussar att de har ett fungerande koncept.

1.2 Syfte

Att ta fram en självständig enhet för duplicering av förarinformation och kommando att använda av lärare vid övningskörning av buss.

1.3 Avgränsningar

Projektet innefattar att ta fram en funktionell produkt för ändamålet, användarvänligheten är i ett första skede nerprioreterat. Slutgiltig paketering av produkten och eventuella förändringar i befintliga bussar kommer inte att tas med.

1.4 Precisering av frågeställningen

Målet med arbetet är att efter genomförandet ha ett koncept för en självständig styrenhet som kan agera som en instrumentpanel. Styrenheten behöver ha funktioner som att kommunicera med bussens system via CAN-nätet, både ta emot samt skicka information till och från körskoleläraren och eleven(föraren). Styrenheten och dess ingående delar skall kunna strömförsörjas från bussen. Hela enheten, inklusive de delar som inte ingår i detta arbete, skall kunna monteras och demonteras på en arbetsdag.
Frågor som skall besvaras under arbetets gång:

- Kan man med enkla medel tillverka en produkt som kan fungera som instrumentpanel till en buss?
- Är en touchskärm ett bra sätt att styra bussens funktioner?
- Vad krävs för att kontrollera funktioner i bussen via CAN?
- Krävs förändring på befintliga bussar för att implementera vår enhet?

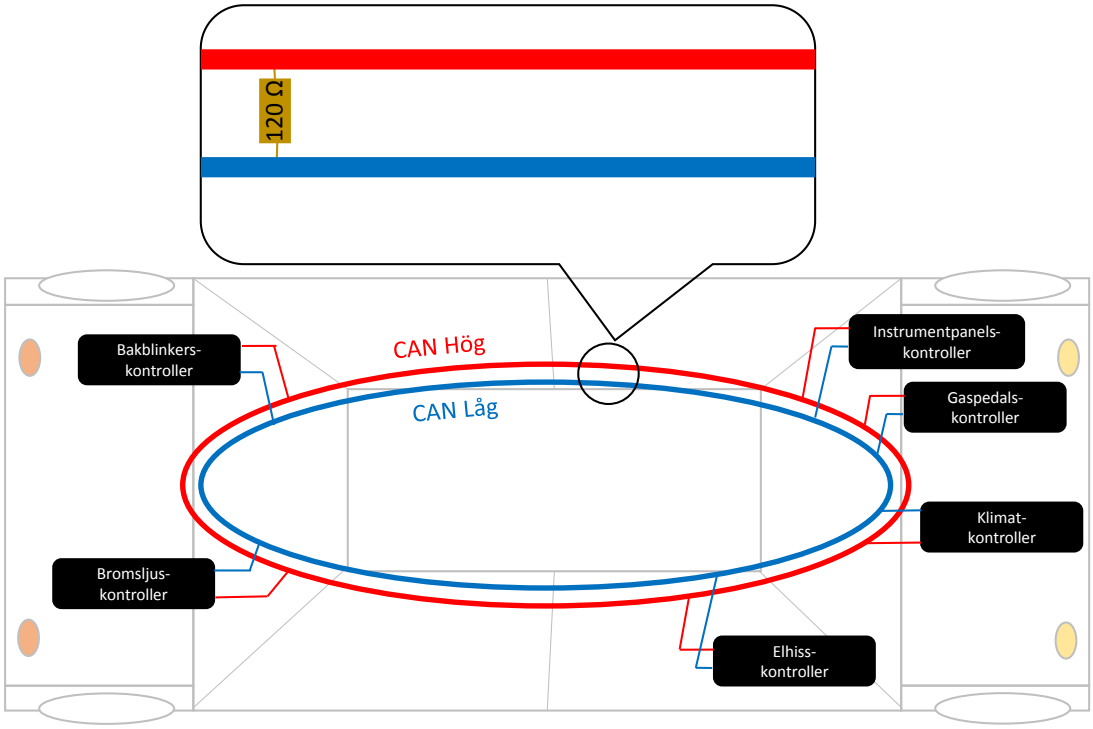
2

Teknisk bakgrund

Detta kapitel sammanfattar den viktigaste tekniska kunskaperna till detta projekt.

2.1 CAN, Controller Area Network

CAN är ett fältbussystem som används i moderna fordon för att kommunicera seriellt mellan olika enheter i fordonet. CAN är utvecklat så att man kan skicka information från många enheter på samma nät utan risk för kollision. Syftet med CAN i fordon är att minska kabeldragningen och underlätta informationsutbyte mellan enheter i fordonet. Ett CAN-nätverk kan innehålla 30 noder, oftast mer än så. Nodernas intresse kan vara väldigt skilda från varandra ändå så hämtar de sin information på samma nät. Det löser man att genom att alla meddelanden har en egen prioritet, när två meddelande läggs ut samtidigt vinner det med högst prioritet och det andra får vänta. [1]



Figur 2.1: *Illustration av ett CAN-nätverk i en personbil.*

Vill man i ett fordon till exempel tuta så trycker man på tutknappen och då skickas det iväg ett meddelande som innehåller information om att tutan skall startas. Den enheten som styr tutan läser då meddelandet och startar tutan. Detta meddelande skickas sedan

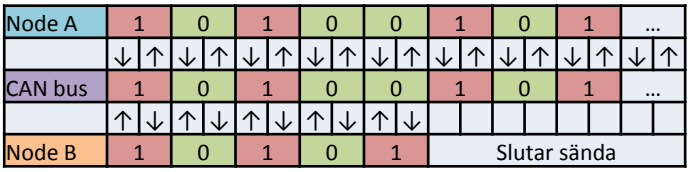
om och om igen med innehållet ”tuta aktiv”. När sedan tutan ska stängas av skickas samma meddelande igen förutom att den innehåller ”tuta inaktiv”. Så funkar ett CAN-meddelande.

2.1.1 Prioritering av meddelande

Om två meddelande vill ut på nätverket samtidigt så skrivs båda ut tills meddelandena skiljer sig åt, då ”vinner” det meddelandet med högst prioritet, det vill säga det med en dominant bit näst på tur. Dominanta bitar på CAN är ”nollor”, vilket betyder att ju lägre värde på identifieraren ett meddelande har desto högre prioritet har det. CANs prioritering funkar precis som en AND-grind mellan minst två sändande noder.

Node A	Node B	CAN bus
0	0	0
0	1	0
1	0	0
1	1	1

Figur 2.2: *Demonstration av dominanta bitar med hjälp av AND-logik.*



Figur 2.3: *När två CAN-meddelanden skickas samtidigt.*

2.1.2 Meddelandets uppbyggnad

Ett meddelande består av upp till 127 bitar, dessa är uppdelade på följande sätt:

1 bit	11 bit	1 bit	1 bit	18 bit	1 bit	6 bit	8-64 bit	15 bit	1 bit	1 bit	1 bit	7 bit
0		1	1		0						1	1
Start of frame	Identifier	SRS	IDE	Extended Identifier	RTR	DLC	Data	CRC	CRC - del	ACK - Slot	ACK - Del	End of frame

Figur 2.4: *Så här är ett CAN-meddelande uppbyggt.*

SOF - Start of frame

Denna bit signalerar att nu kommer ett meddelande. Detta för att synka så att alla mottagande noderna vet när ett meddelande startar.

Identifierare

11 eller 27 bitar som dels bestämmer vilken prioritet meddelandet har när det skickas iväg och viken information meddelandet innehåller. Noderner läser av identifieringsbitarna för att kontrollera om meddelandet innehåller relevant information för just den noden.

RTR - Remote Transmission Request

CAN-meddelanden skickas normalt med jämna intervall, då ettställs inte denna bit. Det är möjligt för en nod att begära data från en annan nod genom att skicka en RTR.

DLC - Data length code

Anger hur många byte information som DATA innehåller.

DATA

Denna del är mellan 8 och 64 bitar stor. Databitarna innehåller information till de berörda noderna som tar emot meddelandet.

CRC - Cyclic Redundancy Check

Kontrollerar om meddelandet är korrekt eller om det har blivit skadat. Alla noder som mottagit ett meddelande beräknar CRC utifrån tidigare del av meddelandet genom att dividera med ett angivet tal och jämföra resten. Resten blir alltså CRC-koden.

CRC -delimiter

Alltid en etta.

ACK-slot

En bit som säger om CRC överensstämmde. Sändande nod lägger ut en etta, om någon av de andra noderna har uppfattat meddelandet korrekt, det vill säga om CRC stämde överens så lägger den eller de andra noder ut en nolla. Nolla är som bekant dominant över etta, på detta sätt vet den sändande noden om någon av noderna uppfattat meddelandet korrekt. De noderna som eventuellt inte uppfattar meddelandet korrekt meddelar detta genom skicka iväg en Error Frame, se avsnitt 2.1.5.

ACK- delimiter

Alltid en etta.

EOF - End of frame

7 ettor som säger att meddelandet är slut.

2.1.3 Överföringsmedia

Ett CAN-nätverk består normalt av tre kablar, CAN hög, CAN låg samt jord. Informationen fås genom att man avläser när potentialen ändras mellan CAN hög och CAN låg. Det finns även CAN-nät som använder sig av andra media, till exempel fiber. I ett CAN-nätverk har inte jord något att göra med informationsflödet, utan används bland annat för att minska risken för störningar.

2.1.4 Bit stuffing

För att alla noder i nätverket skall vara synkroniserade tidsmässigt så synkas de vid flanker i meddelandet, dvs när nästa bit i ett CAN-meddelande inte har samma värde som föregående. Om ett meddelande består av väldigt många lika bitar i rad blir det långt mellan flankerna och noderna kan hamna i otakt. Detta åtgärdas med bit stuffing som innebär att om fem bitar i rad är lika så ändras den sjätte biten till det motsatta värdet och ignoreras av noderna vid tolkning av meddelandet.

2.1.5 Error Frame

En Error Frame skickas iväg om en nod har upptäckt ett fel i överföringen, den består av sex nollor i rad som skickas iväg så fort pågående meddelande är färdigskickat. De fel som leder till en Error Frame är:

- Bit error, en nod försöker att skicka en nolla men läser samtidigt är det en etta på bussen eller tvärtom.
- Stuff error, om sex lika bitar upptäcks på bussen.
- CRC error, om den mottagna CRC-koden inte överensstämmer, se 2.1.2
- Form error, om en bit i CAN-meddelandet som har bestämd status inte har sitt förväntade värde, till exempel en av det fasta ettorna är en nolla.
- Acknowledgement error, om en sändande nod inte besvaras med att övriga noder har mottagit meddelandet korrekt.

2.2 Standarder

2.2.1 ISO

ISO står för International Organization for Standardization och är en oberoende organisation för standardisering[2] Målet för organisationen är att möta marknadens behov av standardisering, bland annat för att underlätta internationellt samarbete. ISO har gjort över 19 500 symboler som används inom en rad olika industrier från tillverkning till medicin. Symboler som binkers, hel/halv-ljus och parkeringsbroms är typer av ISO-standariserade symboler.

2.2.2 SAE

SAE, Society of Automotive Engineers International, är en organisation som utvecklar standarder, de har större delen av sina utvecklade standarder inom fordons- och rymdindustrin.

2.2.3 Placering av symboler och reglage i fordon

Projektet följer ISO 16121 som säger hur informationsanordningar och manöverkontroller skall placeras i fordon. Även då för placering av symboler och reglage på instrumentpanelen. Syftet är att man ska känna igen sig oberoende av vilket fordon man sitter i samt att göra förarmiljön så ergonomisk och bekväm som möjligt. Detta för att inte påfresta föraren mer än nödvändigt varken psykiskt eller fysiskt, standarden bygger på vetenskaplig forskning[3]. Generellt kan man säga att det som är mest relevant, så som hastighet, växel och varv, är placerat rakt framför chaufförens.

2.2.4 CAN-standard

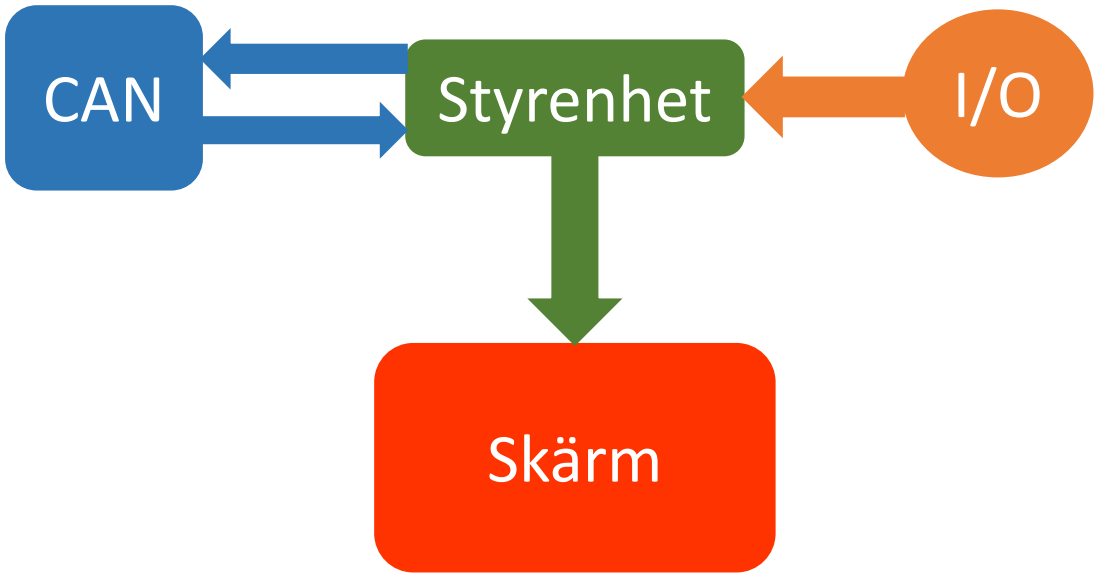
SAE J1939 är en standard för CAN-kommunikation som används av tunga fordon. Standarden omfattar till exempel den fysiska uppbyggnaden av nätet, hur felmeddelanden skall hanteras samt hur meddelanden skall vara uppbyggda. En del meddelanden är helt förbestämda med både identifierare och innehållet i databitarna men det finns också utrymme för varje användare att lägga till egna meddelanden utefter behov. Standarden specificerar också att DLC alltid skall vara 8 bytes[4].

3
Problemformulering

3.1 Kravformulering

Följande punkter och illustration beskriver de krav som uppdragsgivaren ställde på slutprodukten.

- Skriva och läsa på CAN.
- Skärm för att visa åtminstone hastighet, blinkers, varningsblinkers, helljus och parkeringsbroms.
- Interaktiva möjligheter för att kunna skriva på CAN och styra funktioner.
- Signalera när övningsledaren bromsar.



Figur 3.1: *Grafisk beskrivning av produktens uppgifter.*

3.2 Precisering av problem

Utifrån kravformuleringen gjordes en problemformulering som skulle ge ett utkast till vilka problem som behövdes lösas framöver. Dessa frågor ställdes:

- Hur fungerar CAN?

- Vilken hårdvara kan användas?

  Hur ges input till hårdvaran?

  Vilket gränssnitt ska användas för kommunicera mellan styrenheten och CAN-hårdvaran?

- Vilken typ av skärm behövs?

4
Metod

4.1 Arbetsgång

Allt började med att studera liknande projekt för att få en egen uppfattning av projektets omfattning, på den informationen gjordes senare en projektbeskrivning med en utförlig tidsplan. Sedan genomfördes efterforskning och granskning av olika komponenter för att skräddarsy en lösning för det här projektet. Denna informationen hämtades mest genom sökning på internet. Vecka två var konceptet och komponenterna valda då gjordes en beställning. CAN-boarden och skärmen beställdes från utlandet vilket innebar längre leveranstid. I väntan simulerades det i RPi och lästes på om Python. När CAN-boarden kom så började en vecka av att ta emot och skicka meddelanden. Sedermera kom även skärmen och tiden lades på det grafiska gränssnittet. Alla grafiska detaljer gjordes själva med hjälpmedel som Adobe Photoshop och Illustrator. Efter 6 veckor var produkten klar enligt kravspecifikationen plus några extra saker som har lagts till för ökad användarupplevelse.

5
Val av utrustning

För att lösa ut vilka typer av komponenter som kan tänkas ingå i den slutgiltiga produkten så gjordes en sondering av marknaden utifrån kraven på funktionalitet och att komponenterna skulle kunna arbeta tillsammans. Utrustningen som valdes och motiv till valen presenteras i detta kapitel.

5.1 Raspberry Pi

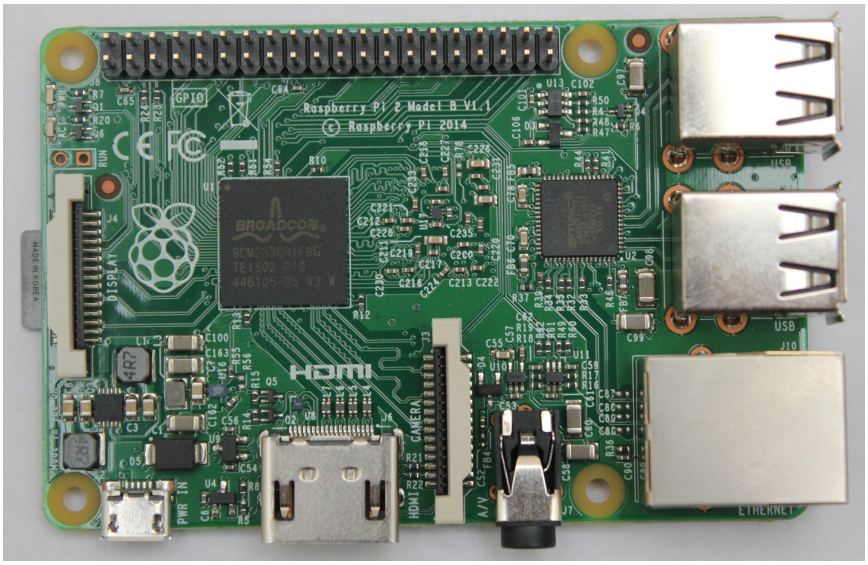
Valet av styrenhet sågs som det viktigaste valet då denna i hög grad skulle påverka valet av övriga komponenter. Den skulle finnas monterad på ett utvecklingskort för att minska arbetsåtgången och vara kompatibel med CAN, skärm och möjlighet till I/O samt ha tillräckligt med processorkraft för att klara av sina uppgifter.

Valet föll på RPi som är en enkortsdator i fickformat. Den skapades från början som ett billigt undervisningsredskap som man lätt kan använda i skolor. RPi har sedan sin lansering används i många olika sammanhang och nu säljs över hälften av alla RPi till kommersiella prudukter[5]. Den är enkel att använda då den använder sig av linux operativsystem. Linux operativsystem har öppen källkod vilket har lett till att utveckling gått väldigt fort framåt eftersom människor och företag har sett dess potential. RPi finns i flera versioner och hårdvaran uppdateras för varje ny version men konceptet är detsamma.

Broadcom BCM2836 är chipet som sitter på RPi 2, modell B som används i detta projekt. Chipet innehåller processor, grafikprocessor, RAM-minne mm.

Funktion	**Raspberry Pi 2**
Processor:	ARM Cortex-A7 900 MHz
Grafikprocessor:	Dual Core VideoCore IV® Multimedia Co-Processor
RAM minne:	1 GB, delat med grafikprocessorn
SPI modul	max 125 MHz
HDMI utgång	ja
Ljudutgång	3.5 mm
Nätverksuttag	ja
USB	4 st USB 2.0
GPIO	40 st
Övrigt	Uttag för kamera och skärm mm

Figur 5.1: *Egenskaper Raspberry Pi 2, modell B.*



Figur 5.2: *Foto på Raspberry Pi.*

RPi valdes för att:
- Den är mycket billig i förhållande till vad den erbjuder i form av hårdvara.
- Den köps klar att använda.
- Den är lätt att konfigurera och komma igång med.
- Välbeprövad produkt.

5.2 CAN-board PICAN

En CAN-board är ett kretskort som består av samtliga komponenter som krävs för gränssnittet mellan CAN-buss och den styrande microkontrollern. I detta projekt används PICAN, en CAN-board speciellt framtagen för RPi.

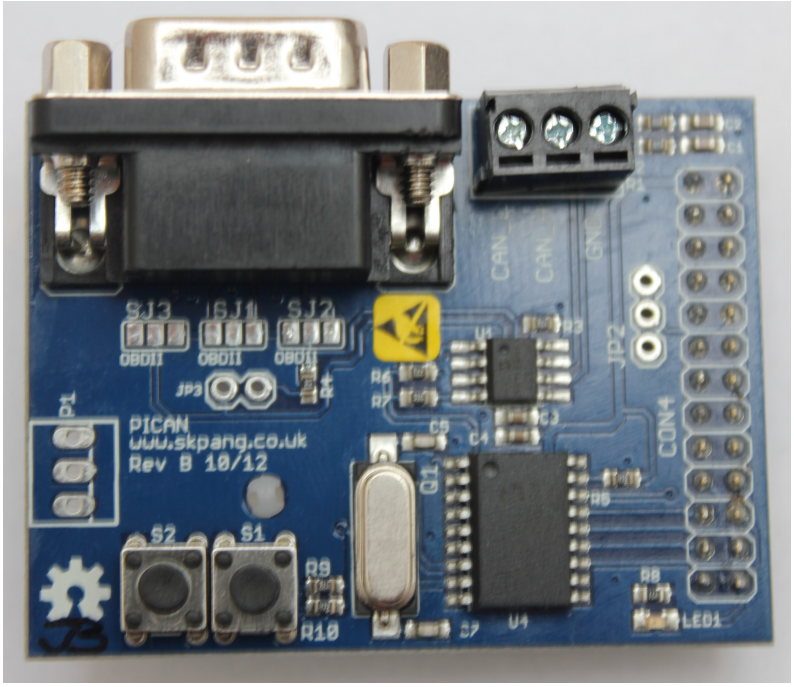
PICAN består i huvudsak av nedanstående komponenter.

5.2.1 CAN-kontroller - MCP2515

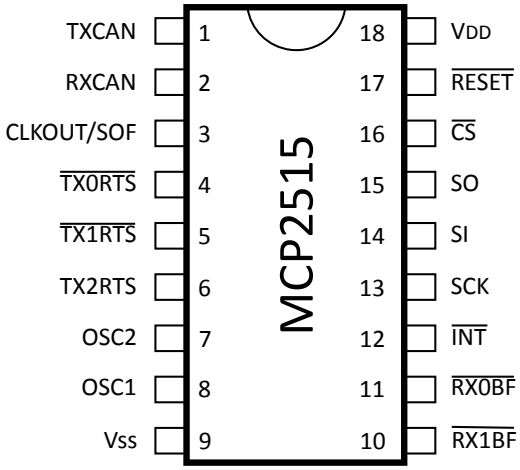
En CAN-kontroller är en liten enhet som sköter hela det logiska gränssnittet från SPI till CAN inklusive felhantering, SOF, EOF mm. På den aktuella CAN-boarden används CAN-kontrollern MCP2515 från Microchip, se figur 5.4. Kontrollern kan hantera både standard frame och extended frame och följer standarden 2.0B med en maxhastighet på 1 Mbit/s. All information om MCP2515 är hämtad ur dess datablad[6].

5.2.2 CAN-transciever - MCP2551

CAN-transcievern sitter kopplad mellan CAN-kontrollern och CAN-bussen för att anpassa signalerna. Den tar också hand om eventuella störningar från CAN-nätet, tex höga spänningstransienter[7].



Figur 5.3: *Foto på CAN-boarden för projektet.*



Figur 5.4: *Pinout för CAN-kontrollern MCP2515.*

5.2.3 Kristall

Boarden har också en oscillatorkristall på 16 MHz för att få en stabil klockfrekvens till CAN-kontrollern.

5.2.4 Externa anslutningar

Till RPi har CAN-boarden anslutningar för jord, spänningsmatning 5 V och 3.3 V, SPI samt ett antal vanliga portar för t.ex. interrupt. CAN-boarden har möjlighet att ansluta till CAN-bussen antingen via ett DB9-uttag eller via lösa kablar till CAN hög och CAN låg samt jord. Ett vanligt sätt att ansluta till CAN-bussen externt till ett fordon är via ODB2, det kan boarden enkelt göra med en adapter mellan DB9 och ODB2.

5.2.5 Motivering av val

PICAN valdes för att:
- Den är anpassad för RPi, dvs ingen anpassning krävdes.
- Den använder MCP2515, en vanlig CAN-kontroller med bra dokumentation.
- Den har ett DB9-uttag.
- Den stödjer CAN 2.0B, vilket var vad som krävdes för projektet.
- Den klarar baudrate upp till 1 Mbit/s vilket är tillräckligt för projektet.
- Den hade tillfredsställande leveranstid.

5.3 Skärm

Skärmen skall fungera som en instrumentpanel till föraren fast i detta fall till övningsledaren och med begränsad funktionalitet. En skärm som skall användas i den här miljön ställs det en del krav på. Dock behövde inte alla uppfyllas, inte förrän det skall tillverkas en färdig produkt. Kraven blev:
- Hög ljusstyrka, för att skärmen ska vara läsbar när det är starkt solljus ute.
- Hög betrakningsvinkel, för att kunna se skärmen från olika sittställningar.
- Drivs på 5V, så att man kan driva den med samma spänningskälla som raspberryn.
- Touchfunktion, för att inte begränsa forsatta utvecklingsmöjligheter.
- Över 7 tum skärm.

Eftersom styrenheten valdes först blev kompatibilitet för skärmen mot styrenheten en avgörande faktor. Efter att ha undersökt marknaden så fanns det egentligen bara ett alternativ som är kompatibel med RPi från början. Skärmen som valdes syns i figur 5.5.

Figur 5.5: *Foto på skärmen.*

6

Hårdvarukonstruktion

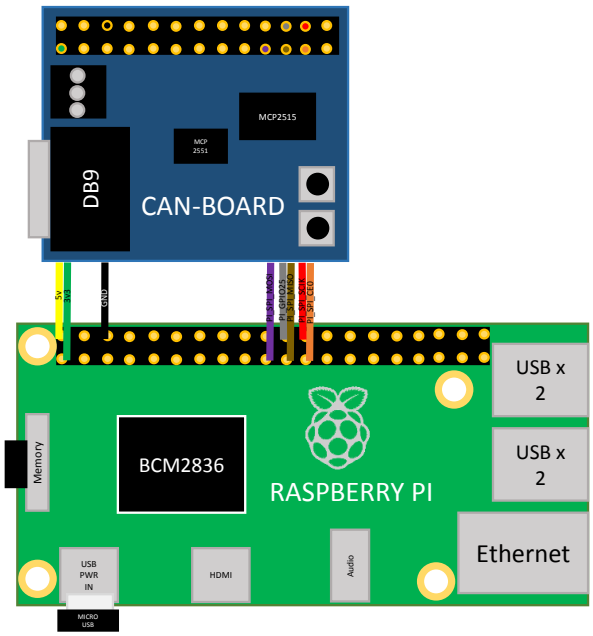
Tack vare att de komponenter som valdes bestod av färdiga kort som redan var kompatibla med varandra så har inte hårdvarukonstruktion varit en stor del av detta projekt. Med alla komponenter bestämda börjar jobbet med att pussla ihop allt till en slutprodukt. Nedan följer hur komponenterna sitter ihop.

6.1 Strömförsörjning

RPi strömförsörjs med 5 V DC genom ett micro-USB uttag, övriga enheter strömförsörjs i sin tur via RPi. Strömförsörjningen har lösts på två olika sätt. Under utvecklingen skedde det med en AC/DC-omvandlare från elnätet (230 V) och för att strömförsörja i fordon används en omvandlare från 12-24 V i cigarettuttaget till micro-USB.

6.2 Sammankoppling Raspberry Pi och CAN-board

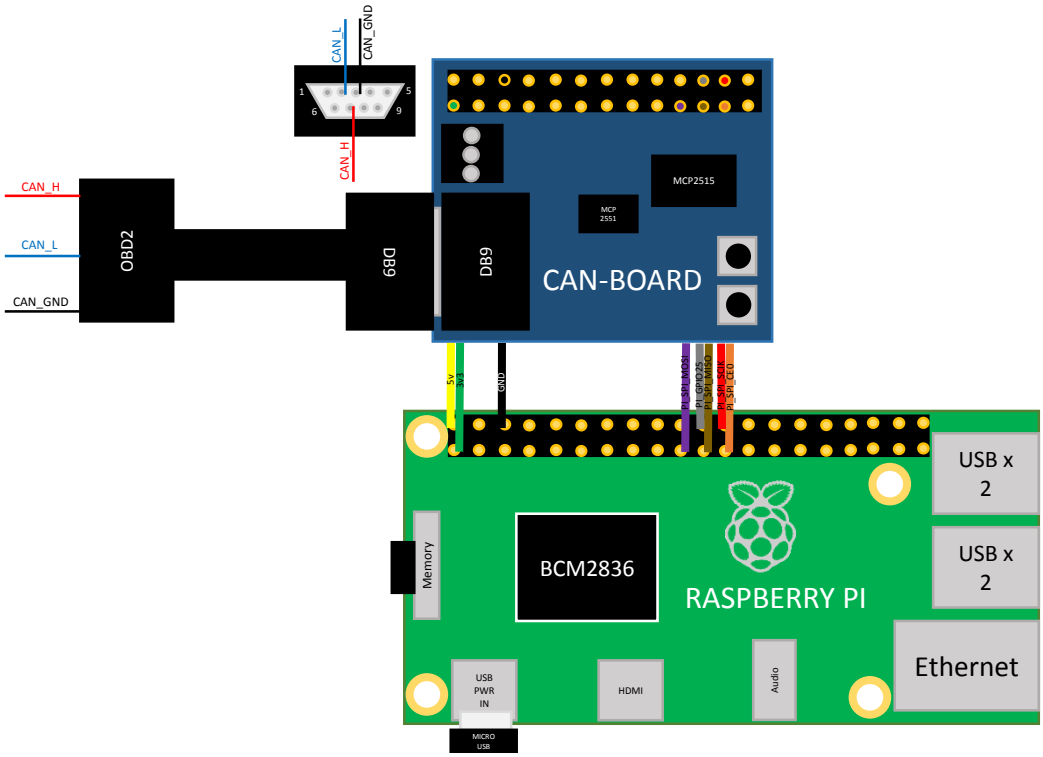
CAN-boarden är konstruerad för att monteras direkt på RPi första 26 GPIO-pinnar. Av dom 26 pinnarna behövdes bara 8 för det här ändamålet. För mer detaljer om inkopplingen se bilden nedan och bilaga 1 (kretsschema för CAN-boarden).



Figur 6.1: *Sammankoppling av Raspberry Pi och CAN-board.*

6.2.1 Koppling till CAN-bussen

För att koppla CAN-boarden till CAN-bussen används en D-subkontakt med 9 pinnar. CAN hög ligger på ben 7, CAN låg på ben 2 och CAN jord på ben 3. För att sedan komma åt fordonets CAN-buss kopplar man t.ex. in i dess serviceuttag genom en OBD2 kontakt.

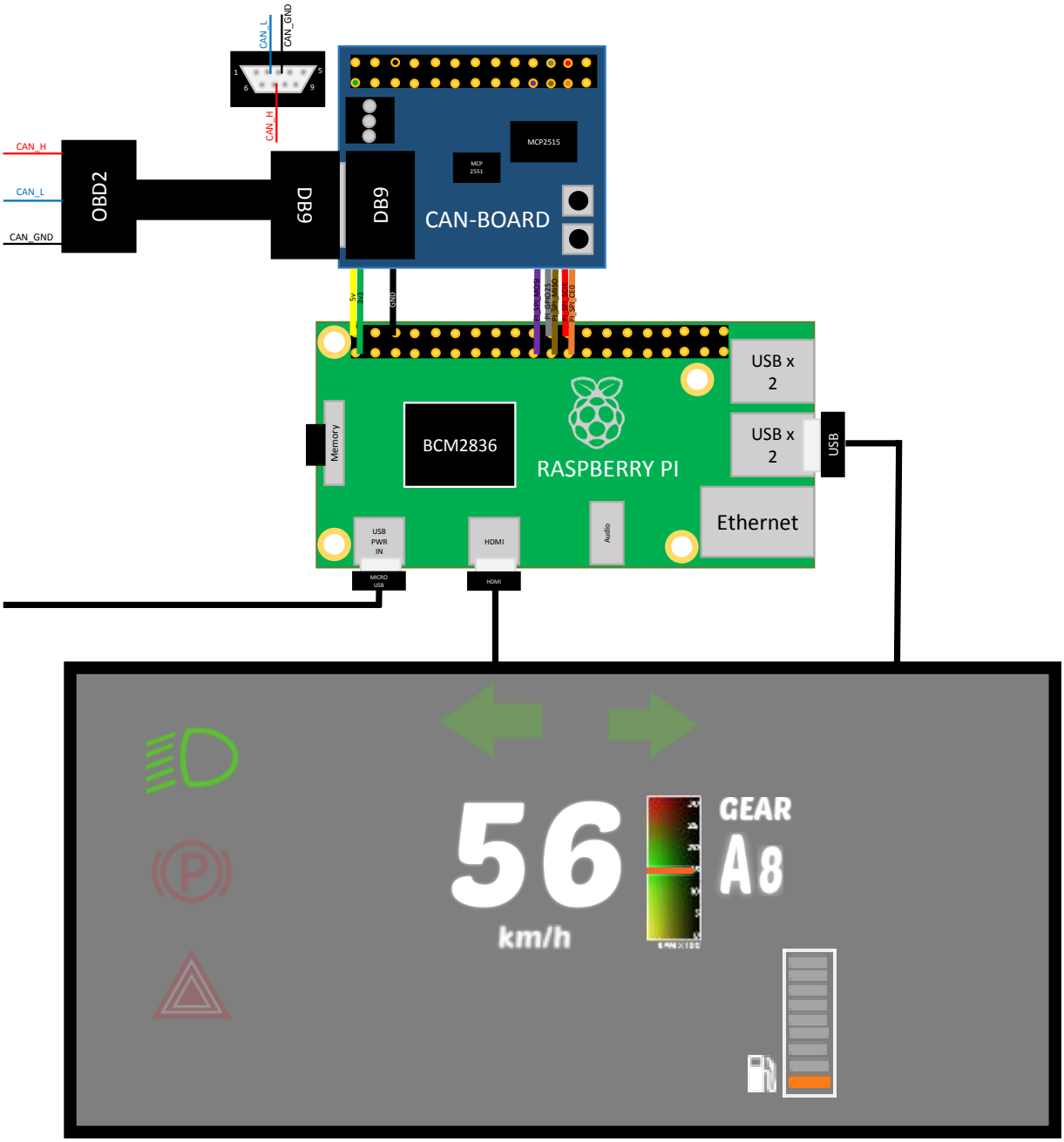


Figur 6.2: *Sammankoppling av Raspberry Pi och CAN-board med CAN-anslutning.*

6.3 Sammankoppling av Raspberry Pi och Skärm

Skärmen kan strömförsörjas genom USB, till en början så räckte inte strömmen från RPi USB-portar till. När skärmen skulle startas så ströps spänningen till den stötvis på RPi vilket gjorde att skärmen blinkade. Detta verifierades med ett oscilloskop. Det gick dock att lösa problemet på ett enkelt sätt. RPi kan leverera 1.2 A genom USB men standardinställnignen är max 0.6 A. För att tillåta RPi att leverera 1.2 A krävdes denna raden i config.txt: ”max_usb_current = 1”.

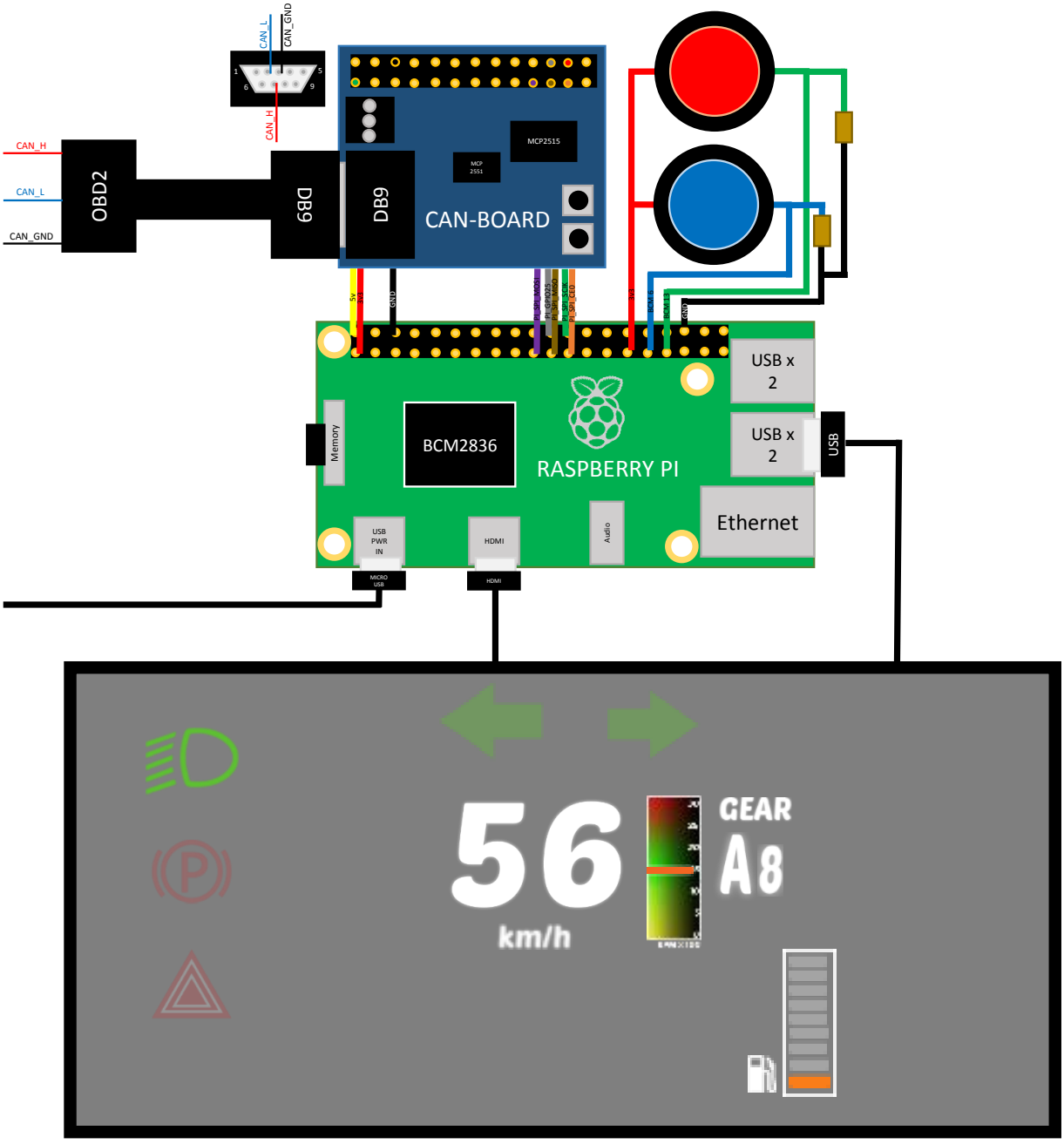
Skärmen ansluts till RPi via HDMI, RPi anpassade sig automatiskt till att skriva ut en upplösning som passade skärmen någorlunda. Men för att få det perfekt krävdes manuella inställningar. I RPi konfigfil (config.txt) finns parametrar som talar om vilken upplösning som skall skickas ut via HDMI. Dessa inställningar valdes :
- hdmi_group = 2 (DMT)
- hdmi_mode = 81
Detta gav upplösningen 1366x768 [8] vilket överensstämmer med skärmen.
Grundinställningen overscan fick tas bort för att få skärmen att visa bild ända ut i kanterna. Efter ändringar i config.txt krävs omstart av systemet för att de nya inställningarna skall börja gälla.



Figur 6.3: *Sammankoppling av Raspberry Pi, CAN-board med CAN-anslutning och skärm.*

6.4 Inkoppling av knappar till Raspberry Pi

För att koppla knappar till RPi användes enklast möjliga sätt. Brytarna kopplades mellan 3.3 V och den GPIO som skulle reagera på knapptrycket, en pulldown-resistor kopplades också mot jord. Kontaktstudsar hanteras av mjukvaran. Knapparna är tänkt att i framtiden användas för att ge komandon till fordonet, till exempel tuta.



Figur 6.4: *Sammankoppling av Raspberry Pi, CAN-board med CAN-anslutning, skärm och knappar.*

7
Mjukvarukonstruktion

7.1 Mjukvaruval

När hårdvaran var vald så skulle den också programmeras, valet av processor gjorde att valet av programmeringsspråk var långt ifrån självklart. På grund av författarnas bakgrund och tidigare erfarenheter så hade det varit naturligt att programmera i programmeringsspråket C. Dock är programmeringsspråket C komplicerat när det kommer till grafiska detaljer som är en del av jobbet. Så en efterforskning på andra programmeringsspråk gjordes. Efterforskningen bestod i att läsa på och testa på olika högnivå-språk som var relevanta för ämnet. De språk som undersöktes var Java, C++ och Python. Det som gjorde att vi fastnade för Python var dess enkelhet och att det är väldigt komplett språk till RPi, det finns färdiga moduler för det mesta.

7.2 Mjukvara

7.2.1 Raspbian

Raspbian är operativsystemet som används på RPi. Det är baserat på Linuxdistrubitionen Debian speciellt utvecklat för RPi. [9] Rasbian innehåller redan från start mycket av de funktioner som man förväntar sig av ett operativsystem till en persondator, som en webbläsare, en filhanterare och mycket annat.

7.2.2 Python

Python är ett förhållandevis sent utvecklat objektorienterat högnivå-programmeringsspråk. Det är känt för sin enkla och lättlästa kod. Har man tidigare bekantat sig med C-språk är Python väldigt lätt att ta till sig. En stor fördel med Python är att det finns en rad olika funktioner som gör grafisk programmering mycket mindre och kompaktare jämfört med C-språket. Python lämpades sig bra för den produkt som skulle skapas och för författarnas inlärningstid.[10]

Pygame

Pygame är en utvecklingsmodul för Python gjord för att kunna skriva spel. Den innehåller flera verktyg som gör grafisk programmering lättare. [12]

Spidev

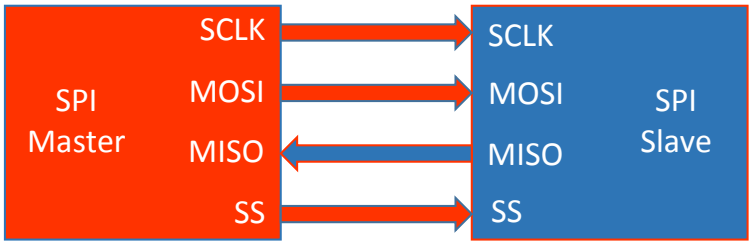
Py-spidev är en Python-modul som använder sig av linuxkärnans befintliga drivrutiner för att styra SPI-modulen på RPi. [13]

För att initiera modulen behövs:

```
import spidev

spi = spidev.SpiDev()
spi.open(0, 0)              #Open spi port 0, device (CS) 0
spi.mode = 0                #Clock polarity and phase [CPOL|CPHA]
spi.max_speed_hz = 10*1000000    #10 MHz (SPI-clock)
spi.bits_per_word = 8
```

7.2.3 Serial Peripheral Interface (SPI)

Kommunikationen mellan CAN-kontrollern och RPi består av SPI, seriell seriekommunikation. SPI är vanligt förekommande inom inbyggda system och består av fyra trådar(se figur 7.1). Detta ger kommunikationen möjlighet att vara ”full duplex”, det betyder att man kan sända och ta emot samtidigt. I denna tillämpning används SPI hastigheten 7.8 MHz. Bakgrunden till det är att maxhastigheten för CAN-kontrollern är 10 MHz och RPi går endast att ställa stegvis, 7.8 MHz är den hastighet närmast under 10 MHz som går att ställa in [14].



Figur 7.1: *Serial Peripheral Interface - SPI,*
SCLK - Serial Clock
MOSI - Master Output Slave Input
MISO - Master Input Slave Output
SS - Slave Select

7.2.4 Konfiguration av MCP2515

All konfiguration av kontrollern sker med hjälp av SPI från RPi under initieringsfasen. Här nedan följer ett urval av de mest relevanta konfigurationerna som gjorts, för ytterligare information se datablad MCP2515[6] eller för att ta reda på hur det gjordes i detta projekt se kod med början på rad 30 i bilaga 6.

- Hastighet på CAN-nätet, kontrollerns CAN-hastighet beror också av vilken oscillatorfrekvens som används vilket måste tagas i beaktan vid konfigureringen. Microchip har ett digitalt hjälpmedel[15] för att ta fram önskad baudrate i MCP2515.
- För att avlasta styrenheten kan ett antal filter och maskar i kontrollern användas för att filtrera bort frames som styrenheten inte är intresserad av. Ett meddelande med samma identifierare som ett filter kommer att accepteras medans masken talar om vilka bitar i ett filter som inte behöver överensstämma för att meddelandet ändå

skall tas emot. Det vill säga masken kan, rätt konfigurerad, göra att flera frames kan tas emot med ett och samma filter. Det finns sex filter och två maskar i MCP2515. I bilaga 2 finns ett exempel på hur filter och maskar fungerar.

- Tala om för kontrollern hur den skall agera när den mottager ett CAN-meddelande.
- Hur den skall göra om den misslyckas med ett att skicka ett meddelande.
- Hur den skall prioritera meddelanden som den skall skicka iväg.

7.2.5 Sändning av CAN-meddelande

Vid varje meddelande som skickas görs följande:

- Bestämmer vilken identifierare som skall användas och om det är en standard identifier eller en extended identifier.
- Bestämmer vilken buffer i kontrollern som skall laddas (det finns tre).
- Laddar buffern med identifierare och erforderligt antal databytes(DLC).
- En Reguest to send (RTS) skickas till kontrollern vilket ger den tillåtelse att så fort CAN-bussen är ledig skicka iväg framen.

Allt detta sker i funktionen send_FRAME se bilaga 6 rad 186.

7.2.6 Mottagande av CAN-meddelande

Vid mottagande av ett meddelande sker följande:

- Kontrollern signalerar att den mottagit ett meddelande genom att dra ner INT-benet till 0 V (se bilaga 1).
- RPi känner med hjälp av interrupt direkt av att ett meddelande mottagits. Den skickar en begäran om status till kontrollern som talar om vilken mottagningsbuffert som innehåller det aktuella meddelandet.
- RPi läser in det mottagna meddelandet.
- RPi sparar undan informationen som finns i meddelandet

Allt detta sker i funktionen read_FRAME se bilaga 6 rad 218.

7.3 Grafisk användargränssnitt

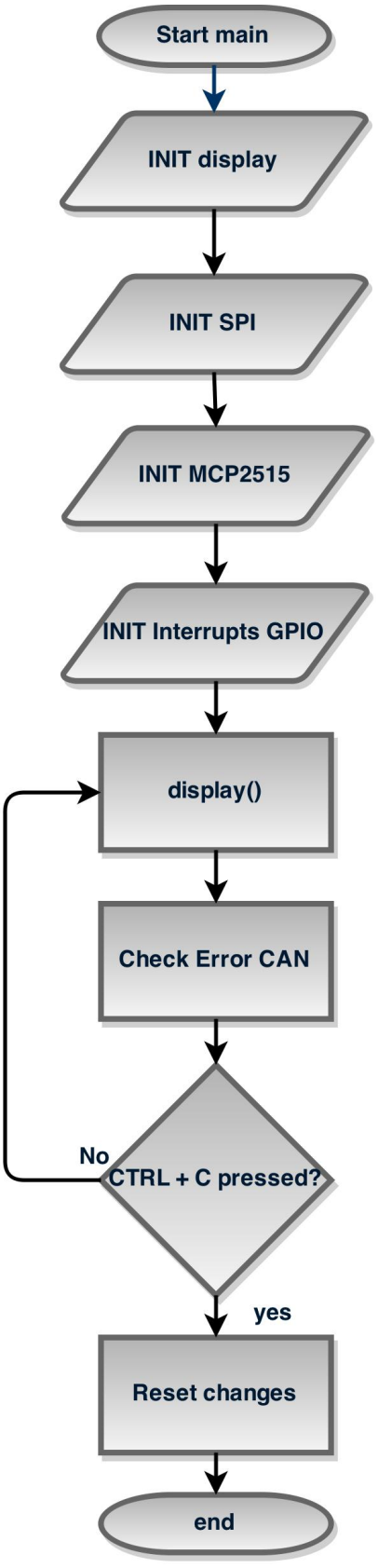
Målet var att få en igenkänningsfaktor mellan en instrumentpanelen för chauffören och skärmen som övningsledaren använder utan att för den sakens skull inte använda de fördelar som kommer med en högupplöst skärm. Placeringen och utformning av symbolerna gjordes efter ISO-standard. Vissa indikatorer har dock uppdaterats för att de skall ge en modernare känsla och för att få produkten att visa på sin stora flexibilitet som framtida produkt. För att ta fram det grafiska gränssnittet har vi använt oss av Pygame och Photoshop.

7.4 Förklaring av programkod

I figur 7.2 presenteras hur huvudprogrammet är uppbyggt. Först sker i vanlig ordning initiering av de i programmet ingående delarna, bilder laddas till det grafiska gränssnittet, SPI konfigureras, CAN-kontrollern konfigureras och RPi ställs in så att interrupt genereras vid önskade händelser.

När programmet körs så tas CAN-meddelanden emot med interrupt och sparar all mottagen information till ”globala” variabler, till exempel hastighet, varvtal och blinkers. Huvudlopen består endast i att köra funktionen display(), samt kolla efter ”error” mellan CAN-kontrollern och RPi. I huvudlopen läggs sedermera send FRAME in för att kunna skicka CAN-meddelanden.

När en knapp trycks ner så genereras en interrupt som ger möjlighet att till exempel ändra en parameter eller köra programkod. All kod och flödesschema utom den kod som innehåller känslig information om CAN-databasen ligger som bilaga. Se bilaga 3-9 för programkod och bilaga 11 för flödesscheman.



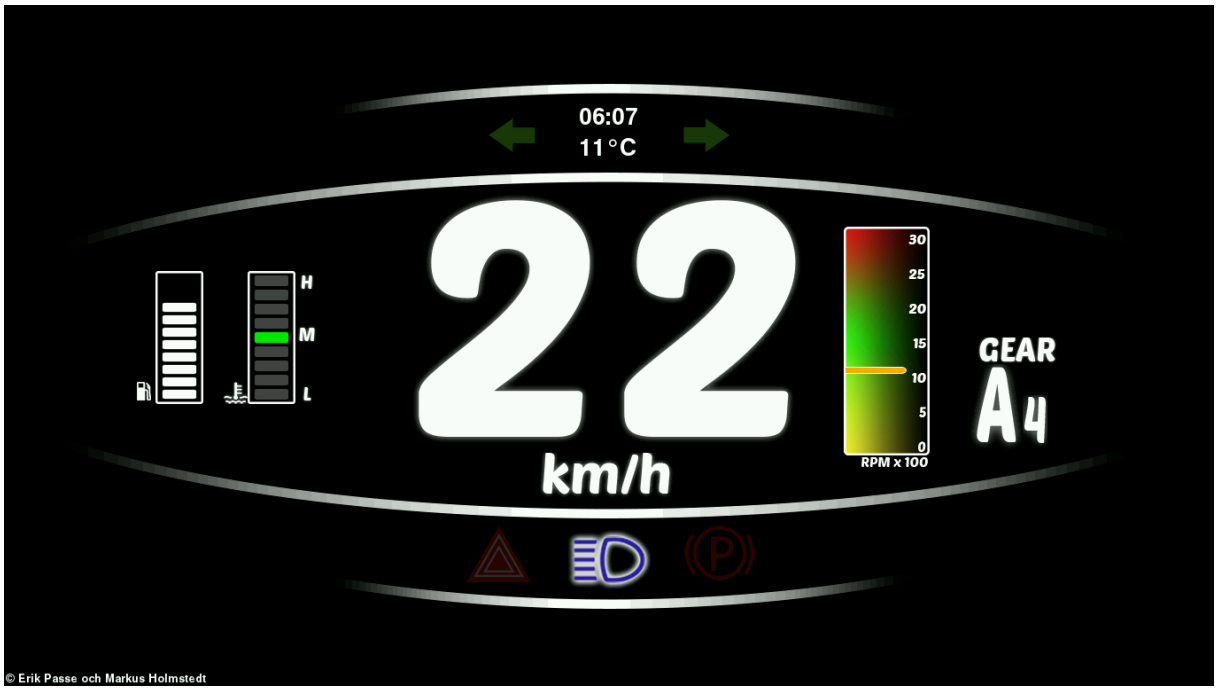
Figur 7.2: *Flödesschema för programmets huvudlop.*

8

Slutprodukt

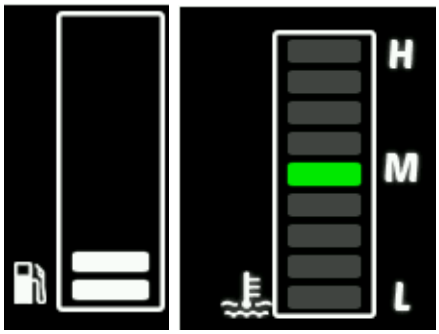
I detta kapitel beskrivs slutprodukten och hur dess funktion testades.

8.1 Beskrivning av slutprodukt



Figur 8.1: *Layout för den framtagna skärmen.*

Figur 8.1 visar en ögonblicksbild från den framtagna skärmgrafiken som visas när produkten är kopplad till ett CAN-nätverk på en körande buss.



Figur 8.2: *Så här mätarna för bränslemängd och kylvätsketemperatur ut.*

Figur 8.2 visar hur bränslemätaren ser ut vid 20 % i bränsletanken och hur mätaren för kylvätsketemperaturen ser ut vid normal motortemperatur. För fler bilder se bilaga 10, där presenteras hur mätarna skiljer sig åt vid olika värden.

8.2 Verifiering av funktion hos slutprodukt

8.2.1 Spegla instrumentpanel

Verifikation av grafiken som visas på skärmen har gjorts i flera steg, dels så testades det under framtagandet av mjukvaran med olika testfall för att se att det som visades på skärmen motsvarade inkommande data. För att verifiera att inkommande data från CAN-nätet beräknas på rätt sätt har en rimlighetsbedömning gjorts under uppspelandet av loggfiler från en buss. Inkoppling mot ett CAN-nätverk löstes genom att spela in loggfiler från en riktig buss med hjälp av CANalyser och sedan spela upp för produkten. En fullständig verifiering för att testa alla möjliga fall av datavärden från CAN-nätet har inte gjorts.

8.2.2 Styra funktioner på bussen

Det normala för en styrenhet i en buss är att styra bussens funktioner direkt via CAN-nätet. Denna funktion har testats i produkten i den mån att CAN-meddelanden kan skickas efter behov och knapptryckningar kan ändra vad som skickas osv.

8.3 Signalera när övningsledare bromsar

Det har lagts väldigt lite tid på denna då den i dagsläget inte tillför så mycket till konceptet. Men föreslagen lösning är att man använder sig av en microbrytare som triggar en piezoelektrisk signalgivare när bromspedalen hos övningsledaren tryckts ner. Vidare föreslås även att den lösning är skild från styrenheten. Alternativ lösning är att koppla microbrytaren till RPi för att då kunna spela ut valfritt ljud.

9
Resultat

Målet har uppnåtts i stort, arbetet har lett till en självständig styrenhet som kan duplicera en instrumentpanel i en buss. Här under svaras på de frågor som ställdes i det inledande arbetet.

Kan man med enkla medel tillverka en produkt som kan fungera som instrumentpanel till en buss?

Produkten är byggd utav komponenter som vem som helst har tillgång till att köpa. Komponentkostanden uppgår till cirka 2000 kr.

Är en touchskärm ett bra sätt att styra bussens funktioner?

Detta arbete har lett fram till att touchskärm inte är en bra lösning när den ska användas i den miljön. Vibrationerna och att man inte kan känna sig fram gör touchskärmen svår att använda när bussen körs. Därför finns idag inga touchfunktioner aktiverade i produkten. Istället finns fysiska knappar som skall fungera som reglage i prototypstadiet.

Vad krävs för att kontrollera funktioner på en buss via CAN?

Man behöver en identitet som en nod i CAN-nätverket. Identiteten innebär att de berörda enheterna på nätet lyssnar på meddelandena som skickas ut från produkten. I detta dupliceringsfall måste även ett prioritetsproblem lösas då det finns två enheter som kan skicka två olika informationer om samma sak samtidigt. Detta problem ligger utanför avgränsningarna i arbetet och är inget som har kunnat påverkas. Utan identitet går det bara att lyssna på nätverket och spegla den information som redan finns där, vilket produkten kan göra idag.

Krävs förändring på befintliga bussar för att implementera vår enhet?

Vår enhet måste få en identitet som befintlig nod i bussen CAN-nätverk. Det kräver att man lägger till våra meddelanden i databasen och anpassar övriga berörda enheter därefter, det vill säga CAN-nätverket måste förberedas för vår enhet.

10
Diskussion

Detta arbete har nått sitt slut och vi anser att produkten visar på potential som är god nog att arbeta vidare med.

10.1 Styrkor

Som koncept så har produkten en styrka i dess enkelhet samt att den går att skala om efter användningsområde, det finns få begränsningar i utvecklingsmöjligheter som ligger inom ramen för vad produkten kan påverka, det vill säga att svårigheten ligger i att de system som produkten skall samarbeta med måste anpassas. Produkten har dessutom låg komponentkostnad.

10.2 Svagheter

Det finns många svagheter i den så här långt framtagna produkten om den skulle användas skarpt, men som proof of concept är den en bit på vägen helt beroende på vilken nivå man lägger ribban på.

10.3 Utvecklingsmöjligheter

För att produkten ska nå slutgiltigt funktionsstadie krävs att man arbetar närmare en slutkund så att krav på produkten blir tydligare. För att fortsätta att förbättra den som koncept kan man:

- Ordna så att den agerar rätt när eventuella fel uppstår på CAN
- Förfina det grafiska gränssnittet.
- Lägga till fler funktioner, till exempel en ”gaspådragsmätare”.
- Förbereda den mer mjukvarumässigt för att integrera aktivt på CAN, inte bara lyssna som den gör i dagsläget.
- Göra den kompatibel med en bil för att kunna visa på funktionalitet där.
- Installera möjligheter för att visa pdf:er från ett USB-minne.
- Koppla ihop den med t.ex. en videokamera, för att kunna visa på det breda användningsområdet.
- Använda touchfunktionen på skärmen för att kunna byta layout eller ändra andra inställningar.
- Förbereda för ljuduppspelnig vid inbromsning från körskolelärare.
- Program som beräknar och visar hur sparsam körningen varit.
- Paketera den i en bättre förpackning.

Ett av produktens problem är att om den skall kunna styra funktioner via CAN så måste bussen anpassas för detta i relativt stor utsträckning. Ett annat sätt för att undvika detta är att elektriskt koppla in sig parallellt på samma knapp som bussen har från början. Det har dock inte genomförts någon undersökning om detta är praktiskt möjligt.

10.4 Saker som kunde gjorts annorlunda

Man kunde ha använt en surfplatta och kommunicera trådlöst med en egen enhet som är kopplad mot CAN, detta hade dock inte varit bra om man vill styra funktioner på bussen. Man kunde ha anpassat den till en bil från början, eftersom det är lättare att få tag på för att sedan när allt fungerar som det ska anpassa för bussar.

Litteraturförteckning

[1] Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice, Marco Di Natale, Haibo Zeng, Paolo Giusto and Arkadeb Ghosal, Springer 2012

[2] ISO, Internationella standardiseringsorganisationen, http://www.iso.org/iso/home/about.htm, (Acc 2015-05-27)

[3] SVENSK STANDARD SS-ISO 16121-3:2011, SWEDISH STANDARDS INSTITUTE, Utgåva 2, Publicerad: 2011-09-23

[4] SVENSK STANDARD, SS-ISO 15765-2:2011, Utgåva 2, SWEDISH STANDARDS INSTITUTE, Publicerad: 2011-11-30

[5] Pyttedatorn Raspberry Pi gör megasuccé, Ny Teknik, Marie Alpman, http://www.nyteknik.se/tekniknyheter/article3868627.ece, (Acc 2015-05-27)

[6] MCP2515, Microchip, 2012-07-08, http://ww1.microchip.com/downloads/en/DeviceDoc/21801G.pdf (Acc 2015-05-27)

[7] MCP2551, Microchip, 2010-09-07, sid 1, http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf (Acc 2015-05-27)

[8] RPiconfig, Video, http://elinux.org/RPiconfig, (Acc 2015-05-27)

[9] Raspbian, https://www.raspbian.org/, (Acc 2015-05-27)

[10] Python, https://docs.python.org/3/tutorial/index.html, (Acc 2015-05-27)

[11] Mark Lutz, ”Learning Python, Fourth Edition”, O’Reilly Media Inc. 2009

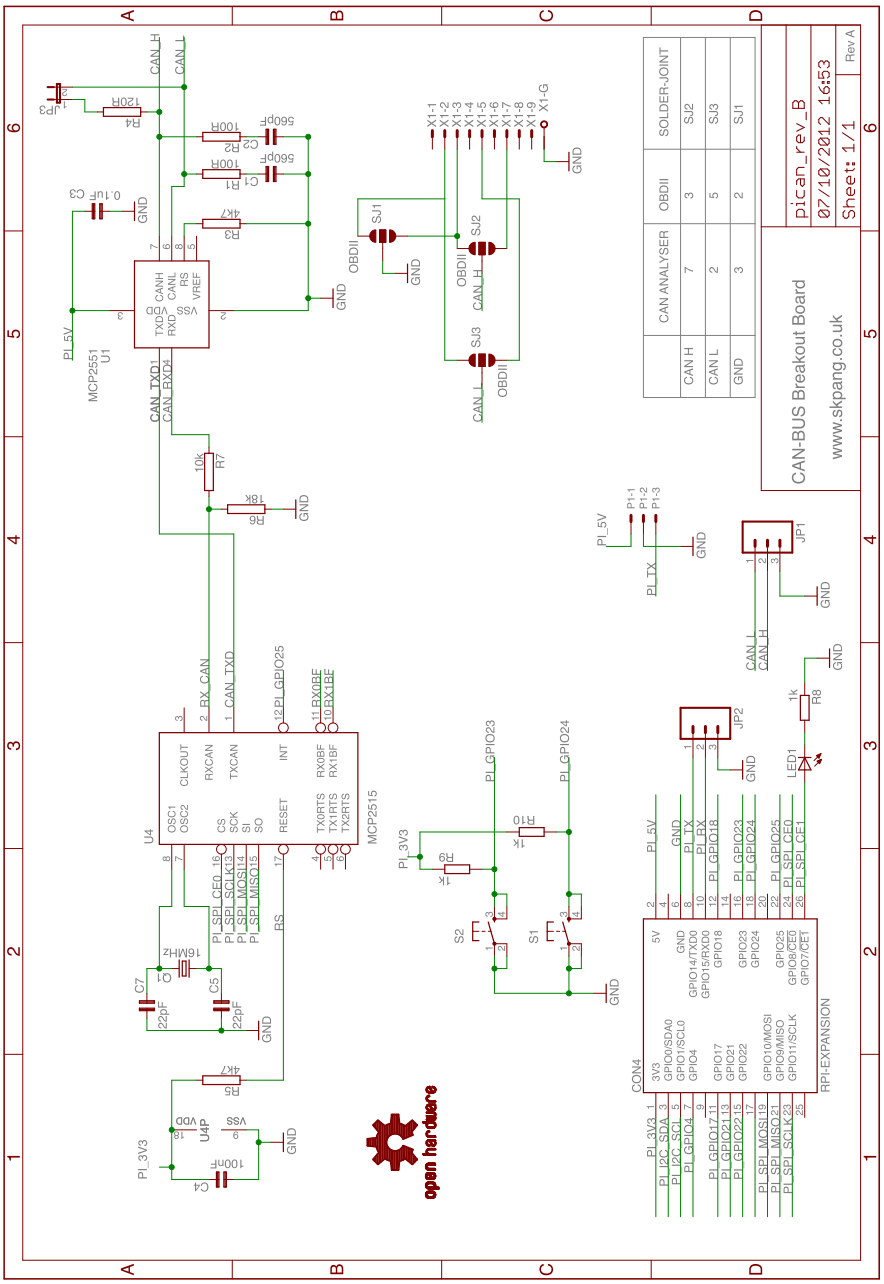
[12] Pygame, modul till Python http://www.pygame.org/wiki/about, (Acc 2015-05-27)

[13] Python Spidev, https://github.com/doceme/py-spidev, (Acc 2015-05-27)

[14] RPi dukumentation om SPI https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md, (Acc 2015-05-27)

[15] Microchip CAN Bit Timing Calculator, Intrepid Control System Inc. http://www.intrepidcs.com/support/mbtime.htm (Acc 2015-05-27)

Bilaga 1 - Kretsschema PICAN



Bilaga 2 - Maskar och filter

Här presenteras hur filter och maskar fungerar i MCP2515:

Mask bit	Filter bit	Message Identifier bit	Accept or Reject bit
0	don't care	don't care	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Om alla bitar i en meddelandes identifierare blir accepterade så godkänns meddelandet. Nedan ges ett exempel med en förkortad identifierare (3 bitar):

Mask	Filter	Message Identifier	Accept or Reject
000	xxx	xxx	Accept
111	101	100	Reject
111	101	101	Accept
xxx	101	101	Accept
001	xx1	xx1	Accept
001	xx1	xx0	Reject
x = don't care bit			

Bilaga 3 - Kod main

```
1  #!/usr/bin/env python
2  #This program is made by Erik Passe and Markus Holmstedt 2015
3  #Display and CAN functionality
4  print("Lets go!")
5
6  from display import*
7  from CAN import*
8  from interruptGPIO import*
9  import time
10
11 time.sleep(1) #Startup delay
12 try:
13         while True:
14                 display()
15                 check_ERROR_CAN()
16
17
18 except KeyboardInterrupt:        # Ctrl+C pressed, so
19         spi.xfer([0b11000000])  # RESET and SET CON FIG mode
20         spi.close()                     # Close the port before
              exit
21         GPIO.cleanup()
22         pygame.quit ()
```

Bilaga 4 - Kod skärm

```
1  #display.py
2  #This is the code to display things on the screen
3  print("display.py imported")
4
5  from displayH import*
6  import globals1
7
8
9  # -------- Main Program Loop -----------
10
11 #Uncomment to view full screen on start
12 #pygame.display.toggle_fullscreen()
13
14 # Used to manage how fast the screen updates
15 clock = pygame.time.Clock()
16
17 def display():
18         global frame_count
19
20         # Set the screen background
21         screen.fill(black)
22
23         #*** Clock ***#
24         clock_posx = ScreenW-font.size("00:00")[0]
25         clock_posy = 0
26         output_string = "{0:02}:{1:02}".format(globals1.
              TD_I_Hours, globals1.TD_I_Minutes)
27         text = font.render(output_string, True, lightblue)
28         screen.blit(text, [clock_posx, clock_posy])
29         #----- end clock -----#
30
31         #*** Speedometer ***#
32         sizeofspeed = pygame.Surface.get_size(Speed_0)
33         posx_speed = (ScreenW-sizeofspeed[0])/2
34         posy_speed = 150
35
36         #Km/h symbol
37         sizeof_textspeed = pygame.Surface.get_size(Speed_text)
38         screen.blit(Speed_text, [(posx_speed+sizeofspeed[0]/2)-
              sizeof_textspeed[0]/2, posy_speed+sizeofspeed[1]])
39
```

```
40          #Speed numbers
41          Speed_num = [Speed_0, Speed_1, Speed_2, Speed_3, Speed_4,
                Speed_5, Speed_6, Speed_7, Speed_8, Speed_9]
42
43          #Calculate smooth speed change to display
44          globals1.VehicleSpeed_list.append(globals1.VehicleSpeed)
45          if len(globals1.VehicleSpeed_list) > 40:
46                  globals1.VehicleSpeed_list.pop(0)                #
                       Remove first item in list if list contains more
                        than 40 elements
47          AveregeVehicleSpeed = sum(globals1.VehicleSpeed_list)/len
                (globals1.VehicleSpeed_list)
48
49          #Left number speed
50          if AveregeVehicleSpeed >= 10:
51                  screen.blit(Speed_num[(AveregeVehicleSpeed/10)
                       %10], [posx_speed-sizeofspeed[0]/2, posy_speed
                       ])
52          else:   #Move right number to center
53                  posx_speed = posx_speed - sizeofspeed[0]/2
54
55          #Right number speed
56          screen.blit(Speed_num[AveregeVehicleSpeed%10], [
                posx_speed+sizeofspeed[0]/2, posy_speed])
57          #----- end speedometer -----#
58
59          #*** Gear indication ***#
60          #Gear text
61          Geartext_posx = 1100
62          Geartext_posy = 350
63          screen.blit(Gear_text, [Geartext_posx, Geartext_posy])
64
65          #Driving mode M,A,N,R
66          Drive_mode_posx = Geartext_posx-25
67          Drive_mode_posy = Geartext_posy + 30
68          Drive_mode = Drive_mode_P
69
70          if globals1.M_N_A_R == 1:
71                  Drive_mode = Drive_mode_R
72          elif globals1.M_N_A_R == 2:
73                  Drive_mode = Drive_mode_N
74          elif globals1.M_N_A_R == 3:
75                  Drive_mode = Drive_mode_A
76          elif globals1.M_N_A_R == 4:
77                  Drive_mode = Drive_mode_M
78          else:
79                  Drive_mode = Drive_mode_E       #Error
80
81          screen.blit(Drive_mode, [Drive_mode_posx, Drive_mode_posy
                ])
82
```

```
83          #Gear 1, 2, 3.. etc
84          Gear_posx= Geartext_posx + 50
85          Gear_posy = Geartext_posy + 65
86          Gear = [GearE, Gear1, Gear2, Gear3, Gear4, Gear5, Gear6,
               Gear7, Gear8, Gear9, Gear10, Gear11, Gear12, GearE,
               GearE, GearE]
87          if globals1.M_N_A_R == 3 or globals1.M_N_A_R == 4: #if
               manual or automatic display gear
88                screen.blit(Gear[globals1.TransCurrentGear], [
                     Gear_posx, Gear_posy])
89          #----- end gear indication -----#
90
91          #*** Fuel gauge ***#
92          Fuel_posx=850
93          Fuel_posy=600
94          screen.blit(Fuel_mount, [Fuel_posx-4, Fuel_posy])       #
               Box
95
96          #Fuel dots
97          fuel_level=(int(globals1.FuelLevel)+5)/10       #
               0,1,2,3,4,5
98
99          fuel_dot_posy= Fuel_posy+132
100         Deltaplupp=-14  #Step between dots
101
102         for i in range(1,fuel_level):   #White dots above pos 1
103               screen.blit(Fuel_dot_white, [Fuel_posx, (
                     fuel_dot_posy + i*Deltaplupp)])
104
105         if fuel_level == 1: #Pos 1
106               screen.blit(Fuel_dot_orange, [Fuel_posx,
                     fuel_dot_posy])
107         elif fuel_level >= 2:
108               screen.blit(Fuel_dot_white, [Fuel_posx,
                     fuel_dot_posy])
109
110         #Fuel symbol
111         if fuel_level>1:
112               Fuel_symbol_color = Fuel_symbol_wite
113         else:
114               Fuel_symbol_color = Fuel_symbol_orange
115
116         screen.blit(Fuel_symbol_color, [Fuel_posx-25, Fuel_posy
               +125])
117         #----- end fuel gauge -----#
118
119         #*** Engine coolant temperatur ***#
120         Coolant_posx = 700
121         Coolant_posy = 600
122         Coolant_dot_posy = Coolant_posy+131
123         Delta_dot_Coolant = -16  #Distance between dots
```

```
124         sizeof_Coolant_mount = pygame.Surface.get_size(
               EngineCoolant_mount)
125         sizeof_Coolant_symbol = pygame.Surface.get_size(
               EngineCoolant_symbol)
126
127         screen.blit(EngineCoolant_mount, [Coolant_posx,
               Coolant_posy])  #Box
128         screen.blit(EngineCoolant_symbol, [Coolant_posx-
               sizeof_Coolant_symbol[0], Coolant_posy +
               sizeof_Coolant_mount[1]-sizeof_Coolant_symbol[1]])
               #Symbol for coolant temp
129
130         #Unactive dots coolant temp
131         for i in range(0,9):
132                 screen.blit(EngineCoolant_dot_white_unactive, [
                       Coolant_posx + 4, (Coolant_dot_posy + i *
                       Delta_dot_Coolant)])
133
134         #Active dots coolant temp
135         EngineCoolant_dots = [EngineCoolant_dot_1,
               EngineCoolant_dot_2, EngineCoolant_dot_3,
               EngineCoolant_dot_4, EngineCoolant_dot_5,
               EngineCoolant_dot_6, EngineCoolant_dot_7,
               EngineCoolant_dot_8, EngineCoolant_dot_9]
136         Coolant_dot_posy = Coolant_posy+131
137         Delta_dot_Coolant = -16 #Distance between dots
138
139         if globals1.EngineCoolantTemp >= 38 and globals1.
               EngineCoolantTemp < 128 :       #deg C
140                 Actual_coolant_dot = int((globals1.
                       EngineCoolantTemp-38)/10)            # 0-8
141                 screen.blit(EngineCoolant_dots[Actual_coolant_dot
                       ], [Coolant_posx + 4, (Coolant_dot_posy +
                       Actual_coolant_dot * Delta_dot_Coolant)])
142         #----- end engine coolant temperatur -----#
143
144         #*** Indicators ***#
145         Ind_posx = ScreenW/2
146         Ind_posy = 25
147         Space_Indicators = 50
148         Ind_size = Symbol_indicators_left_active.get_rect().size
149
150         #Indicate left symbol
151         if globals1.LeftFlash:
152                 screen.blit(Symbol_indicators_left_active, [
                       Ind_posx-Ind_size[0]-Space_Indicators,Ind_posy
                       ])
153         else:
154                 screen.blit(Symbol_indicators_left_un_active, [
                       Ind_posx-Ind_size[0]-Space_Indicators,Ind_posy
                       ])
```

```
155
156         #Indicate right symbol
157         if globals1.RightFlash:
158                 screen.blit(Symbol_indicators_right_active, [
                       Ind_posx + Space_Indicators, Ind_posy])
159         else:
160                 screen.blit(Symbol_indicators_right_un_active, [
                       Ind_posx + Space_Indicators,Ind_posy])
161         #----- end indicators -----#
162
163         #*** Left bar ***#
164         #Background
165         bar_posx = 200
166         bar_posy = 0
167         screen.blit(bar, [bar_posx, bar_posy])
168
169         bar_size = pygame.Surface.get_size(bar)
170         bar_symbol_posx = bar_size[0]/2 + bar_posx
171         bar_symbol_delta_y = 100
172
173         #Headlights
174         Beam_symbol_size = pygame.Surface.get_size(
              Symbol_high_beam)
175         Beam_posx = bar_symbol_posx - Beam_symbol_size[0]/2 +5
176         Beam_posy = bar_symbol_delta_y
177         if globals1.HighBeam == 1:
178                 beam = Symbol_high_beam
179         else:
180                 beam = Symbol_low_beam
181         screen.blit(beam, [Beam_posx, Beam_posy])
182
183         #Parking break indicator
184         Parkbreake_symbol_size = pygame.Surface.get_size(
              Symbol_Parkbreak_active)
185         Park_posx = bar_symbol_posx -Parkbreake_symbol_size[0]/2
186         Park_posy = 2 * bar_symbol_delta_y
187         if globals1.ParkingBrakeStatus:
188                 screen.blit(Symbol_Parkbreak_active, [Park_posx,
                       Park_posy])
189         else:
190                 screen.blit(Symbol_Parkbreak_un_active, [
                       Park_posx, Park_posy])
191
192         #Hazard
193         Hazard_symbol_size = pygame.Surface.get_size(
              Symbol_hazard_active)
194         Hazard_posx = bar_symbol_posx -Hazard_symbol_size[0]/2
195         Hazard_posy = 3 * bar_symbol_delta_y
196         if globals1.Hazard:
197                 screen.blit(Symbol_hazard_active, [Hazard_posx,
                       Hazard_posy])
```

```
198         else:
199                 screen.blit(Symbol_hazard_un_active, [Hazard_posx
                        , Hazard_posy])
200         #----- end left bar -----#
201
202         #*** Tachometer ***#
203         Tachometer_posx=950
204         Tachometer_posy=180
205         screen.blit(Tachometer_background, [Tachometer_posx-2,
                Tachometer_posy])        #Background
206
207         #Calculate smooth Tachografshaftspeed changes to display
208         globals1.Tachografshaftspeed_list.append(globals1.
                Tachografshaftspeed) #0-3000 rpm      #Add last rpm to
                end of list
209
210         if len(globals1.Tachografshaftspeed_list) > 20:          #
                Remove first item in list if list contains more than 20
                 elements
211                 globals1.Tachografshaftspeed_list.pop(0)
212
213         Averege_Tachografshaftspeed = sum(globals1.
                Tachografshaftspeed_list)/len(globals1.
                Tachografshaftspeed_list) #Calculate average
                Tachografshaftspeed
214         Tachometer_level = int(0.078*Averege_Tachografshaftspeed
                -240)
215         screen.blit(Tachometer_indicator, [Tachometer_posx  , (
                Tachometer_posy - Tachometer_level)]) #pointer
216         #----- end tachometer -----#
217
218         #*** Door indicator ***#
219         sizeof_door_open = pygame.Surface.get_size(Door_all_open)
220         door_open_posx = ScreenW - sizeof_door_open[0]-10
221         door_open_posy = ScreenH - sizeof_door_open[1]-10
222
223         if (globals1.clicks%3) == 1:
224                 screen.blit(Door_front_open, [door_open_posx  ,
                        door_open_posy])
225         elif (globals1.clicks%3) == 2:
226                 screen.blit(Door_all_open, [door_open_posx  ,
                        door_open_posy])
227         #----- end door indicator -----#
228
229         # Framerate
230         clock.tick(frame_rate)
231
232         # Update the screen with what we've drawn.
233         pygame.display.flip()
```

Bilaga 5 - Inkluderingsfil Skärm

```
1  #displayH.py
2  #This file contains variables and stuff to suport display code
3  print("displayH.py imported")
4
5  import pygame
6
7  # Define some colors
8  black           =(   0,   0,   0)
9  white           =( 255, 255, 255)
10 green           =(   0, 255,   0)
11 red             =( 255,   0,   0)
12 blue            =(100,  109, 251)
13 lightblue       =(0,162,232)
14 yellow  =(246,251,118)
15 varmgrey=(215,210,203)
16
17 # Set the height and width of the screen
18 ScreenH = 768
19 ScreenW = 1366
20 size = [ScreenW, ScreenH]
21
22 ### Initiate ###
23 pygame.init()
24 pygame.font.init()
25 pygame.mixer.quit()
26
27 screen = pygame.display.set_mode(size)
28
29
30 pygame.display.set_caption("Dubbelkommando")
31
32
33 font = pygame.font.Font(None, 40)
34 font1 = pygame.font.SysFont("freesans", 350)
35 frame_count = 0
36 frame_rate = 18
37
38 #*** Load pictures ***#
39 Symbol_indicators_left_active   = pygame.image.load('symbol/
     Symbol_indicators_left_active.png').convert_alpha()
40 Symbol_indicators_left_un_active = pygame.image.load('symbol/
```

```
    Symbol_indicators_left_un_active.png').convert_alpha()
41  Symbol_indicators_right_active  = pygame.image.load('symbol/
    Symbol_indicators_right_active.png').convert_alpha()
42  Symbol_indicators_right_un_active = pygame.image.load('symbol/
    Symbol_indicators_right_un_active.png').convert_alpha()
43  Symbol_high_beam        = pygame.image.load('symbol/
    Symbol_high_beam.png').convert_alpha()
44  Symbol_low_beam = pygame.image.load('symbol/Symbol_low_beam.png')
    .convert_alpha()
45  Symbol_Parkbreak_active = pygame.image.load('symbol/
    Symbol_Parkbreak_active.png').convert_alpha()
46  Symbol_Parkbreak_un_active      = pygame.image.load('symbol/
    Symbol_Parkbreak_un_active.png').convert_alpha()
47  Symbol_hazard_un_active         = pygame.image.load('symbol/
    Symbol_hazard_un_active.png').convert_alpha()
48  Symbol_hazard_active            = pygame.image.load('symbol/
    Symbol_hazard_active.png').convert_alpha()
49  Gear_text       = pygame.image.load('driving mode/
    driving_mode_text.png').convert_alpha()
50  Drive_mode_D= pygame.image.load('driving mode/driving_mode_D.png'
    ).convert_alpha()
51  Drive_mode_N= pygame.image.load('driving mode/driving_mode_N.png'
    ).convert_alpha()
52  Drive_mode_R= pygame.image.load('driving mode/driving_mode_R.png'
    ).convert_alpha()
53  Drive_mode_P= pygame.image.load('driving mode/driving_mode_P.png'
    ).convert_alpha()
54  Drive_mode_A= pygame.image.load('driving mode/driving_mode_A.png'
    ).convert_alpha()
55  Drive_mode_M= pygame.image.load('driving mode/driving_mode_M.png'
    ).convert_alpha()
56  Drive_mode_E= pygame.image.load('driving mode/driving_mode_Error.
    png').convert_alpha()
57  GearE           = pygame.image.load('driving mode/
    driving_mode_gear_Error.png').convert_alpha()
58  Gear1           = pygame.image.load('driving mode/
    driving_mode_gear_1.png').convert_alpha()
59  Gear2           = pygame.image.load('driving mode/
    driving_mode_gear_2.png').convert_alpha()
60  Gear3           = pygame.image.load('driving mode/
    driving_mode_gear_3.png').convert_alpha()
61  Gear4           = pygame.image.load('driving mode/
    driving_mode_gear_4.png').convert_alpha()
62  Gear5           = pygame.image.load('driving mode/
    driving_mode_gear_5.png').convert_alpha()
63  Gear6           = pygame.image.load('driving mode/
    driving_mode_gear_6.png').convert_alpha()
64  Gear7           = pygame.image.load('driving mode/
    driving_mode_gear_7.png').convert_alpha()
65  Gear8           = pygame.image.load('driving mode/
    driving_mode_gear_8.png').convert_alpha()
```

```
66 Gear9           = pygame.image.load('driving mode/
      driving_mode_gear_9.png').convert_alpha()
67 Gear10          = pygame.image.load('driving mode/
      driving_mode_gear_10.png').convert_alpha()
68 Gear11          = pygame.image.load('driving mode/
      driving_mode_gear_11.png').convert_alpha()
69 Gear12          = pygame.image.load('driving mode/
      driving_mode_gear_12.png').convert_alpha()
70
71 bar = pygame.image.load('symbol/Symbol_hallare_170.png').
      convert_alpha()
72
73 EngineCoolant_mount = pygame.image.load('enginetemp/
      enginetemp_dot_holder.png').convert_alpha()
74 EngineCoolant_symbol = pygame.image.load('enginetemp/
      enginetemp_symbol.png').convert_alpha()
75
76 EngineCoolant_dot_white_unactive = pygame.image.load('enginetemp/
      enginetemp_dot_white_un_active.png').convert_alpha()
77 EngineCoolant_dot_1 = pygame.image.load('enginetemp/
      enginetemp_dot_1.png').convert_alpha()
78 EngineCoolant_dot_2 = pygame.image.load('enginetemp/
      enginetemp_dot_2.png').convert_alpha()
79 EngineCoolant_dot_3 = pygame.image.load('enginetemp/
      enginetemp_dot_3.png').convert_alpha()
80 EngineCoolant_dot_4 = pygame.image.load('enginetemp/
      enginetemp_dot_4.png').convert_alpha()
81 EngineCoolant_dot_5 = pygame.image.load('enginetemp/
      enginetemp_dot_5.png').convert_alpha()
82 EngineCoolant_dot_6 = pygame.image.load('enginetemp/
      enginetemp_dot_6.png').convert_alpha()
83 EngineCoolant_dot_7 = pygame.image.load('enginetemp/
      enginetemp_dot_7.png').convert_alpha()
84 EngineCoolant_dot_8 = pygame.image.load('enginetemp/
      enginetemp_dot_8.png').convert_alpha()
85 EngineCoolant_dot_9 = pygame.image.load('enginetemp/
      enginetemp_dot_9.png').convert_alpha()
86
87 Speed_text = pygame.image.load('hastighet/Hastighet_KMH.png').
      convert_alpha()
88 Speed_0 = pygame.image.load('Speed/Speed_0.png').convert_alpha()
89 Speed_1 = pygame.image.load('Speed/Speed_1.png').convert_alpha()
90 Speed_2 = pygame.image.load('Speed/Speed_2.png').convert_alpha()
91 Speed_3 = pygame.image.load('Speed/Speed_3.png').convert_alpha()
92 Speed_4 = pygame.image.load('Speed/Speed_4.png').convert_alpha()
93 Speed_5 = pygame.image.load('Speed/Speed_5.png').convert_alpha()
94 Speed_6 = pygame.image.load('Speed/Speed_6.png').convert_alpha()
95 Speed_7 = pygame.image.load('Speed/Speed_7.png').convert_alpha()
96 Speed_8 = pygame.image.load('Speed/Speed_8.png').convert_alpha()
97 Speed_9 = pygame.image.load('Speed/Speed_9.png').convert_alpha()
98
```

```
99  Fuel_mount       = pygame.image.load('fuel/Fuel_dot_holder.png').
       convert_alpha()
100 Fuel_symbol_wite = pygame.image.load('fuel/Fuel_pump_white.png').
       convert_alpha()
101 Fuel_symbol_orange = pygame.image.load('fuel/Fuel_pump_orange.png
       ').convert_alpha()
102 Fuel_dot_white = pygame.image.load('fuel/Fuel_dot_white.png').
       convert_alpha()
103 Fuel_dot_orange = pygame.image.load('fuel/Fuel_dot_orange.png').
       convert_alpha()
104 Fuel_dot_red = pygame.image.load('fuel/Fuel_dot_red.png').
       convert_alpha()
105 Tachometer_background = pygame.image.load('Tachometer/
       Tachometer_background.png').convert_alpha()
106 Tachometer_indicator = pygame.image.load('Tachometer/
       Tachometer_indicator.png').convert_alpha()
107
108 Door_all_open = pygame.image.load('Door/Door_all_open.png').
       convert_alpha()
109 Door_front_open = pygame.image.load('Door/Door_front_open.png').
       convert_alpha()
```

Bilaga 6 - Kod CAN

```
1  #CAN.py
2  #This is the code to give the program CAN functionality via
       MCP2515
3  print("CAN.py imported")
4
5  from CANH import*
6  import time
7  import globals1
8  import FMID
9
10 #Gives the register value for the given extended identifier
11 #[ID=[ID, DATALENGTH]]
12 def ID_to_registervalues_Extended_Identifier(ID):
13         ID0 = ID[0]>>21
14         ID1 = ((ID[0]>>13)&0b11100000) + ((ID[0]>>16)&0b00000011)
               +8 #+8 for extended frame
15         ID2 = (ID[0]>>8)&0xFF
16         ID3 = ID[0]&0xFF
17         ID4 = ID[1]
18         ID_registervalue = [ID0, ID1, ID2, ID3, ID4]
19         return ID_registervalue
20
21 #Gives the register value for the given filter identifier
22 def ID_to_registervalues_Filter(Filtervalue):
23         ID0 = Filtervalue>>21
24         ID1 = ((Filtervalue>>13)&0b11100000) + ((Filtervalue>>16)
              &0b00000011) +8 #for only apply to extended frame
25         ID2 = (Filtervalue>>8)&0xFF
26         ID3 = Filtervalue&0xFF
27         Filter_registervalue = [ID0, ID1, ID2, ID3]
28         return Filter_registervalue
29
30 def init_MCP2515():
31         WRITE_SPI       = 0b00000010
32         READ_SPI        = 0b00000011
33         CAN_BAUD        = 250   #KHz
34         print("CAN board initialized")
35         spi.xfer([0b11000000])    #RESET and SET CON FIG mode
36         resp = spi.xfer2([READ_SPI, 0x0E, 0x00])        #Read
              CANSTAT Configuration mode =0b1xxxxxxx
37         if (resp[2] & 128) != 0:
```

```
38                print("CANBOARD CONFIGURATION MODE")
39
40        #Init CNF1-CNF3
41        if CAN_BAUD == 125:
42                CNF1 = 0b00000011
43                CNF2 = 0b10111000
44                CNF3 = 0b00000101
45        elif CAN_BAUD == 250:
46                CNF1 = 0b00000001
47                CNF2 = 0b10111000
48                CNF3 = 0b00000101
49        else:
50                print("Felaktig CAN baud")
51        print("CAN_BAUD:" + str(CAN_BAUD))
52        spi.xfer([0b00000010, 0x28, CNF3, CNF2, CNF1])
53
54        #Init TXRTSCTRL
55        TXRTSCTRL = 0b00000000
56        spi.xfer([WRITE_SPI, 0x0D, TXRTSCTRL])  ##
57
58        #Init BFPCTRL
59        BFPCTRL = 0b00000000
60        spi.xfer([WRITE_SPI, 0x0C, BFPCTRL])    ##
61
62        #Init CANINTE -Interrupt enable
63        CANINTE = 0b00000011
64        spi.xfer([WRITE_SPI, 0x2B, CANINTE])    ## Interrubts on
             full RXB 0-1
65
66        #Init TXB0CTRL
67        TXB0CTRL = 0b00000011 ##highest prio
68        spi.xfer([WRITE_SPI, 0x30, TXB0CTRL])   ##
69
70        #Init TXB1CTRL
71        TXB1CTRL = 0b00000010 ##high prio
72        spi.xfer([WRITE_SPI, 0x40, TXB1CTRL])   ##
73
74        #Init TXB2CTRL
75        TXB2CTRL = 0b00000000 ##Lowest prio
76        spi.xfer([WRITE_SPI, 0x50, TXB2CTRL])   ##
77
78        #Init RXB0CTRL  (Obs changed in init_FILTER_MASKS)
79        RXB0CTRL = 0b01100000 ##Recevie all messege, rollover
             DISABLED
80        spi.xfer([WRITE_SPI, 0x60, RXB0CTRL])   ##
81
82        #Init RXB1CTRL
83        RXB1CTRL = 0b01100000 ##Recevie all messege
84        spi.xfer([WRITE_SPI, 0x70, RXB1CTRL])   ##
85
86        #Clear all interruptflags
```

```
87          CANINTF = 0b00000000
88          spi.xfer([WRITE_SPI, 0x2C, CANINTF])
89
90          #Initiate Controller to filter out unwanted messegees
91          init_FILTER_MASKS()
92
93          #Init CANCTRL
94          CANCTRL = 0b00000000    #Normal mode, CLOCKOUT = off,
               disable one shot mode
95          spi.xfer([WRITE_SPI, 0x0F, CANCTRL])
96          print("CANBOARD NORMAL MODE")
97
98  #Put CAN controller in loopback mode
99  def loopback_mode():
100         WRITE_SPI = 0b00000010
101         CANCTRL = 0b01000000    #Loopback mode, CLOCKOUT = off,
               disable one shot mode
102         spi.xfer([WRITE_SPI, 0x0F, CANCTRL])
103         print("LOOPBACKMODE")
104
105
106
107
108 #Initiate Controller to filter out unwanted messegees
109 def init_FILTER_MASKS():
110         WRITE_SPI       = 0b00000010
111         READ_SPI        = 0b00000011
112
113         #RXB0
114         Filter0 = ID_to_registervalues_Filter(FMID.FILTER0) #RXB0
115         Filter1 = ID_to_registervalues_Filter(FMID.FILTER1) #RXB0
116         Mask0 = ID_to_registervalues_Filter(FMID.MASK0)          #
               RXB0
117
118         #RXB1
119         Filter2 = ID_to_registervalues_Filter(FMID.FILTER2) #RXB1
120         Filter3 = ID_to_registervalues_Filter(FMID.FILTER3) #RXB1
121         Filter4 = ID_to_registervalues_Filter(FMID.FILTER4) #RXB1
122         Filter5 = ID_to_registervalues_Filter(FMID.FILTER5) #RXB1
123         Mask1 = ID_to_registervalues_Filter(FMID.MASK1)          #
               RXB1
124
125
126         #Write filter register
127         FILTER = [Filter0, Filter1, Filter2, Filter3, Filter4,
               Filter5]
128         FILTER_number = 0
129         for FILTER_adr in [0x00 ,0x04, 0x08, 0x10, 0x14, 0x18]:
130                 spi.xfer2([WRITE_SPI, FILTER_adr, FILTER[
                       FILTER_number][0], FILTER[FILTER_number][1],
                       FILTER[FILTER_number][2], FILTER[FILTER_number
```

```
                    ][3]]) #Gar att snabba upp denna skrivningen
                       genom att bara skriva som en lista
131                 FILTER_number += 1
132
133         #Write mask register
134         MASK = [Mask0, Mask1]
135         for MASK_num in range(0,2):
136                 spi.xfer2([WRITE_SPI, (0x20 + (MASK_num*4)), MASK
                       [MASK_num][0], MASK[MASK_num][1], MASK[MASK_num
                       ][2], MASK[MASK_num][3]])
137
138         #Activate filter and masks
139         #Init RXB0CTRL
140         RXB0CTRL = 0b00000000 ##Recevie all valid messege(FILTER
               and MASKS = TRUE), rollover enabled
141         spi.xfer([WRITE_SPI, 0x60, RXB0CTRL])   ##
142
143         #Init RXB1CTRL
144         RXB1CTRL = 0b00000000 ##Recevie all valid messege(FILTER
               and MASKS = TRUE)
145         spi.xfer([WRITE_SPI, 0x70, RXB1CTRL])   ##
146
147         print("#Filters and Masks ON")
148
149 #Run this if interrupt in RPi and Can controller are out of step
150 def sync_PI_CANcontroller():
151         WRITE_SPI = 0b00000010
152         READ_SPI        = 0b00000011
153         #Reset interrupts
154         CANINTF=spi.xfer2([READ_SPI, 0x2C, 0x00])
155         spi.xfer2([WRITE_SPI, 0x2C, CANINTF[2]&0b11111100])
156         print("No frames recived, maybe Error in interrupt
               between PI and CAN controller")
157
158
159 #Transfer DATA from int to list of bytes
160 def DATA_to_bytes(DATA, LENGTHOFDATA):
161         if LENGTHOFDATA == 1:   #DLC=1
162                 return [DATA]   #Make DATA a list
163         if LENGTHOFDATA == 2:   #DLC=2
164                 DATA1 = (DATA & 0xFF00)>>8
165                 DATA0 = (DATA & 0xFF)
166                 DATA = [DATA1, DATA0]
167                 return DATA
168         elif LENGTHOFDATA == 3: #DLC=3
169                 DATA2 = (DATA & 0xFF0000)>>16
170                 DATA1 = (DATA & 0xFF00)>>8
171                 DATA0 = (DATA & 0xFF)
172                 DATA = [DATA2, DATA1, DATA0]
173                 return DATA
174         elif LENGTHOFDATA == 4: #DLC=4
```

```
175                 DATA3 = (DATA)>>24      #Look here if add more
                        than 4 bytes of data
176                 DATA2 = (DATA & 0xFF0000)>>16
177                 DATA1 = (DATA & 0xFF00)>>8
178                 DATA0 = (DATA & 0xFF)
179                 DATA = [DATA3, DATA2, DATA1, DATA0]
180                 return DATA
181         else:
182                 print("For manga bytes, bygg ut DATA_to_bytes
                        funktionen" )
183                 return
184
185 #Send FRAME
186 def send_FRAME(ID ,DATA):
187         WRITE_SPI       = 0b00000010
188         READ_SPI        = 0b00000011
189
190         #Choose buffer
191         TXBnCTRL_address = 0x30
192         for x in range(0,3):
193                 test = spi.xfer2([READ_SPI, TXBnCTRL_address, 0
                        x00])
194                 if (test[2] & 0b00001000) == 0:
195                         break
196                 else:
197                         TXBnCTRL_address += 0x10
198
199         #Fill identifyer and data register
200         LISTofRegistervalues= [WRITE_SPI, TXBnCTRL_address+1,
               IDentifyer[ID][0], IDentifyer[ID][1], IDentifyer[ID
               ][2], IDentifyer[ID][3], IDentifyer[ID][4]]
201
202         #Add data to list to send via SPI
203         if IDentifyer[ID][4] >= 1:      #DATA is a list of bytes
204                 LISTofRegistervalues.extend(DATA_to_bytes(DATA,
                        IDentifyer[ID][4])) #IDentifyer[ID][4] =
                        DATALENGTH in bytes
205
206         #Load the FRAME in the selected buffer
207         spi.xfer2(LISTofRegistervalues)
208
209         #Reguest to send (RTS)
210         if TXBnCTRL_address == 0x30:    #Transmitt buffer 0
211                 spi.xfer2([0b10000001])
212         elif TXBnCTRL_address == 0x40:  #Transmitt buffer 1
213                 spi.xfer2([0b10000010])
214         elif TXBnCTRL_address == 0x50:  #Transmitt buffer 2
215                 spi.xfer2([0b10000100])
216
217 #This function is called by GPIO interrupt when CAN bord recived
       messege
```

```
218 def read_FRAME(GPIO_INTerrupt_CAN):
219         WRITE_SPI       = 0b00000010
220         READ_SPI        = 0b00000011
221         DATA0_recieved=-1
222         DATA1_recieved=-1
223         ID0_recived=-1
224         ID1_recived=-1
225 
226         #READ status instruction
227         STATUS = spi.xfer2([0b10100000, 0x00])
228 
229         #Read from loaded buffers#
230         #RXbuffer 0
231         if (STATUS[1]&0b00000001) == 0b00000001:
232                 RXBn_adress = 0x66      #RXB0
233 
234                 #Get ID
235                 ID0=spi.xfer2([READ_SPI, RXBn_adress-5, 0x00, 0
                       x00, 0x00, 0x00, 0x00])
236                 if (ID0[3]&0b00001000): #Extended identifier
237                         ID0_recived= ID0[5] + (ID0[4]<<8) + ((ID0
                           [3]&0b00000011)<<16) + ((ID0[3]&0
                           b11100000)<<13) + (ID0[2]<<21)
238                 else:                                   #Standard
                        Identifier
239                         ID0_recived =(ID0[2]<<3) + ((ID0[3]&0
                           b11100000)>>5)
240 
241                 #Get data (after added number of bytes to recive)
242                 DATALENGHT = [0x00]*(ID0[6]&0b1111)     #Create a
                        list of bytes equal to DCL (data code length)
243                 GETDATA_SPI = [READ_SPI, RXBn_adress]   #List to
                       send to recive data
244                 GETDATA_SPI.extend(DATALENGHT)                  #
                       Adding number of bytes to recive
245                 DATA0 = spi.xfer2(GETDATA_SPI)                  #
                       Reciving data
246 
247                 DATA0_recieved = DATA0[2 : ((ID0[6]&0b1111)+2)]
248 
249         #READ status instruction (again, maybe not necessary)
250         STATUS= spi.xfer2([0b10100000, 0x00])
251 
252         #Rxbuffer 1
253         if (STATUS[1]&0b00000010) == 0b00000010:
254                 RXBn_adress = 0x76      #RXB1
255 
256                 #Get ID
257                 ID1=spi.xfer2([READ_SPI, RXBn_adress-5, 0x00, 0
                       x00, 0x00, 0x00, 0x00])
258                 if (ID1[3]&0b00001000): #Extended identifier
```

```
259                         ID1_recived= ID1[5] + (ID1[4]<<8) + ((ID1
                               [3]&0b00000011)<<16) + ((ID1[3]&0
                               b11100000)<<13) + (ID1[2]<<21)
260                 else:                                       #Standard
                        Identifier
261                         ID1_recived =(ID1[2]<<3) + ((ID1[3]&0
                               b11100000)>>5)
262
263                 #Get data (after added number of bytes to recive)
264                 DATALENGHT = [0x00]*(ID1[6]&0b1111)             #
                       Create a list of bytes equal to DLC
265                 GETDATA_SPI = [READ_SPI, RXBn_adress]   #List to
                       send to recive data
266                 GETDATA_SPI.extend(DATALENGHT)                  #
                       Adding number of bytes to recive
267                 DATA1 = spi.xfer2(GETDATA_SPI)                  #
                       Reciving data
268                 DATA1_recieved = DATA1[2 : ((ID1[6]&0b1111)+2)]
269
270         #If no messege found
271         if (STATUS[1]&0b00000011) == 0b00000000:
272                 print("RX_buffer EMPTY")
273
274
275         #Uncomment to print recived message in terminal
276         #if ID0_recived != -1:
277         #       print("CAN-meddelande mottaget:\nID: " + hex(
                   ID0_recived) + " Data: " + str(DATA0_recieved) + "(
                   decimalt)")
278         #if ID1_recived != -1:
279         #       print("ID: " + hex(ID1_recived) + " Data: " + str
                   (DATA1_recieved) + "(decimalt)")
280         #print(time.time())
281         #print("___")
282
283         if ID1_recived == FMID.TC01_V:  #Hastighet och varvtal
284                 globals1.Tachografshaftspeed = int(((
                       DATA1_recieved[5]<<8) + DATA1_recieved[4])
                       *0.125)  #Varvtal
285                 globals1.VehicleSpeed = int(((DATA1_recieved
                       [7]<<8) + DATA1_recieved[6])*0.00390625)    #
                       Hastighet
286
287         if ID0_recived == FMID.TD_I:    #Tid, Sec, Min, Hour
288                 globals1.TD_I_Seconds = DATA0_recieved[0]
289                 globals1.TD_I_Minutes = DATA0_recieved[1]
290                 globals1.TD_I_Hours = DATA0_recieved[2]
291
292         if ID1_recived == FMID.VP6: # P,N,R,D  LeftFlash,
                   RightFlash, HighBeam, Hazard
293                 FMID.M_N_A_R = (DATA1_recieved[0]>>4) #4:M, 3:A ,
```

```
                    2:N, 1:R
294
295         if ID0_recived == FMID.VP37:    #LeftFlash, RightFlash,
               HighBeam, Hazard
296                 globals1.LeftFlash = ((DATA0_recieved[0]>>4)&0
                       b00000001)
297                 globals1.RightFlash = ((DATA0_recieved[0]>>2)&0
                       b00000001)
298                 globals1.HighBeam = ((DATA0_recieved[1]>>6)&0
                       b00000011)
299                 globals1.Hazard = ((DATA0_recieved[1]>>4)&0
                       b00000001)
300
301         if ID1_recived == FMID.ETC2:    #Current gear
302                 globals1.TransCurrentGear = DATA1_recieved[3]&0
                       b00001111        #Lite oklara innehall i detta
                       meddelande
303
304         if ID0_recived == FMID.VP68:    #Parkingbrake
305                 globals1.ParkingBrakeStatus = ((DATA0_recieved
                       [0])>>4)&1
306
307         if ID1_recived == FMID.VP6:     # M,N,A,R
308                 globals1.M_N_A_R = DATA1_recieved[0]&0b00001111
309
310         if ID1_recived == FMID.ET1:             #Coolant and
               engineoil temperature
311                 globals1.EngineCoolantTemp = DATA1_recieved[0]-40
312
313         if ID0_recived == FMID.AIR1:
314                 globals1.BreakAir1 = DATA0_recieved[2]*8        #
                       Signalen visar bara FF
315                 globals1.BreakAir2 = DATA0_recieved[3]*8        #
316                 globals1.PneumaticSupplyPressure = DATA0_recieved
                       [0]*8
317                 #print("Pressure:" + str(globals1.
                       PneumaticSupplyPressure) + "KPa")
318
319
320
321         #Reset error counter
322         globals1.ERROR_Count = 0
323         #Reset interrupts
324         CANINTF=spi.xfer2([READ_SPI, 0x2C, 0x00])
325         spi.xfer2([WRITE_SPI, 0x2C, CANINTF[2]&0b11111100])
326
327 def check_ERROR_CAN():
328         globals1.ERROR_Count += 1               ##Reset in read
               frame
329         if globals1.ERROR_Count>3:
330                 sync_PI_CANcontroller()
```

```
331
332 #Init SPI
333 spi = spidev.SpiDev()
334 spi.open(0, 0)                          # open spi port 0, device
        (CS) 0
335 spi.mode = 0
336 spi.max_speed_hz = 10*1000000   #10 MHz (SPI-clock)
337 spi.bits_per_word = 8
338
339 #Calculate registervalues for each identifier
340 IDentifyer = [ID_to_registervalues_Extended_Identifier(
       BLINKERS_ID), ID_to_registervalues_Extended_Identifier(
       HEL_HALV_LJUS_ID), ID_to_registervalues_Extended_Identifier(
       P_BROMS_ID), ID_to_registervalues_Extended_Identifier(TUTA_ID)]
        #Blinkers, #Hel/halvljus #P-broms #Tuta
341
342 init_MCP2515()
```

Bilaga 7 - Inkluderingsfil CAN

```
1  #CANH.py
2  #This file contains variables and stuff to suport CAN code
3  print("CANH.py imported")
4
5  import spidev #SPI
6
7  #Extended IDentifiers [ID, DATALENGTH]
8  BLINKERS_ID = [0x15555555, 0]   #Dessa ar for att senda an sa
       lange, gor om senare
9  HEL_HALV_LJUS_ID = [0xB53232, 1]
10 P_BROMS_ID = [0x10757272, 2]
11 TUTA_ID = [0x1935B0B0, 4]
```

Bilaga 8 - Variabler

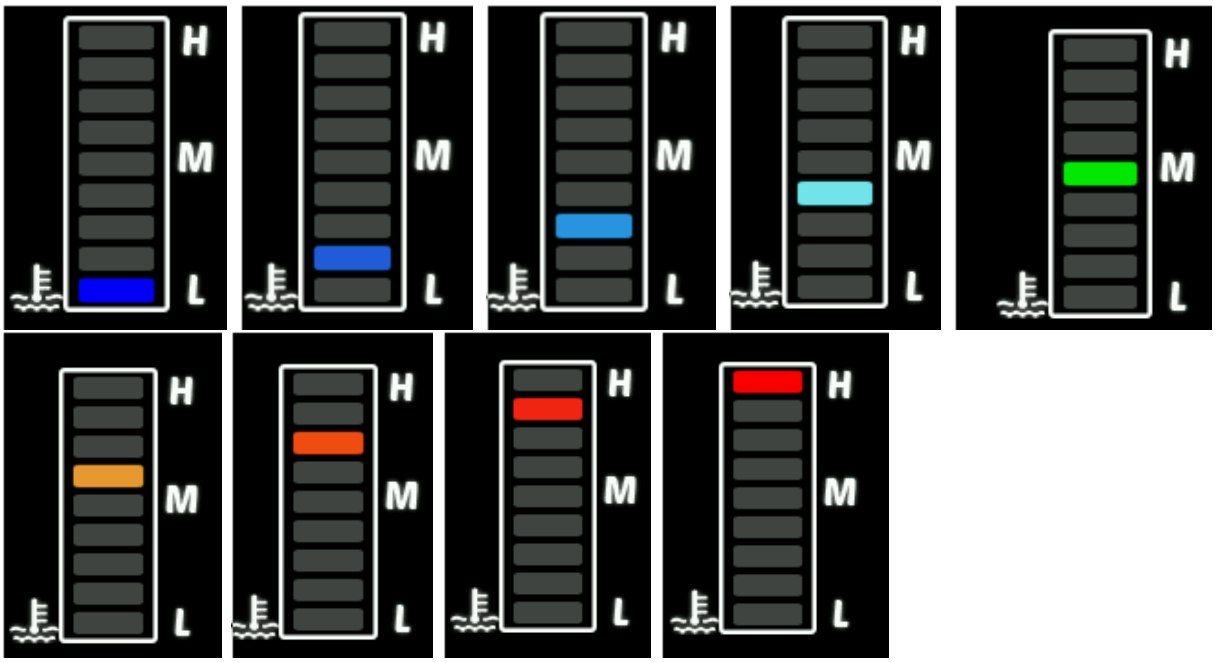
```
#globals.py
#global shared variables
#This file contains all variables shared between display, CAN, and interruptGPIO
print("globals.py imported")

#GLOBAL variables

ERROR_Count=0

#ON_or_OFF = 0
clicks=0

TD_I_Seconds = 59
TD_I_Minutes = 59
TD_I_Hours = 23

M_N_A_R = -1

LeftFlash = 1
RightFlash = 1
HighBeam = 0
Hazard = 1

Tachografshaftspeed = 0
Tachografshaftspeed_list = []

FuelLevel=100           #In percent
VehicleSpeed=-1
VehicleSpeed_list=[]
TransCurrentGear=0      #Default 0 for Error

ParkingBrakeStatus = 1

EngineCoolantTemp = 120
EngineOilTemp = -1

PneumaticSupplyPressure = 0
BreakAir1 = 0
BreakAir2 = 0
```

Bilaga 9 - Kod interrupt

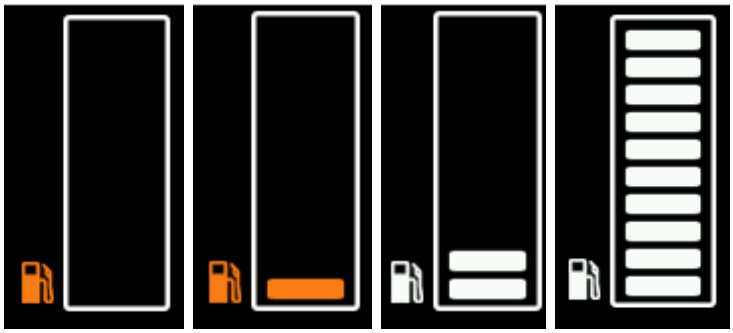
```
1  #interruptGPIO.py
2  #This code gives interrupt and call functions on GPIO interrupt
3  print("interruptGPIO.py imported")
4
5  # get the GPIO Library
6  import RPi.GPIO as GPIO
7  import globals1
8  import CAN
9  import pygame
10
11 #GPIO select
12 GPIO_ON_test = 23
13 GPIO_OFF_test = 24
14 GPIO_INTerrupt_CAN = 25
15
16 #GPIO mode
17 GPIO.setmode(GPIO.BCM)
18
19 #INPUTS
20 GPIO.setup(GPIO_ON_test, GPIO.IN)
21 GPIO.setup(GPIO_OFF_test, GPIO.IN)
22 GPIO.setup(GPIO_INTerrupt_CAN, GPIO.IN) #Interrupt recieved
      messege etc
23
24 #*** Interrupthantering ***#
25 #Test off
26 def OFF(GPIO_OFF_test):
27         globals1.ON_or_OFF=0
28         globals1.clicks +=1
29         pygame.display.toggle_fullscreen()
30
31 #Test on
32 def ON(GPIO_ON_test):
33         globals1.ON_or_OFF=1
34
35 #*** Interruptevent I/O ***#
36 GPIO.add_event_detect(GPIO_OFF_test, GPIO.FALLING, callback=OFF,
      bouncetime=200)        #Test
37 GPIO.add_event_detect(GPIO_ON_test, GPIO.FALLING, callback=ON)
            #Test
38 GPIO.add_event_detect(GPIO_INTerrupt_CAN, GPIO.FALLING, callback=
```

```
    CAN.read_FRAME)          #Read CAN-RB 0-1
```

Bilaga 10 - Bilder på skärmgrafiken

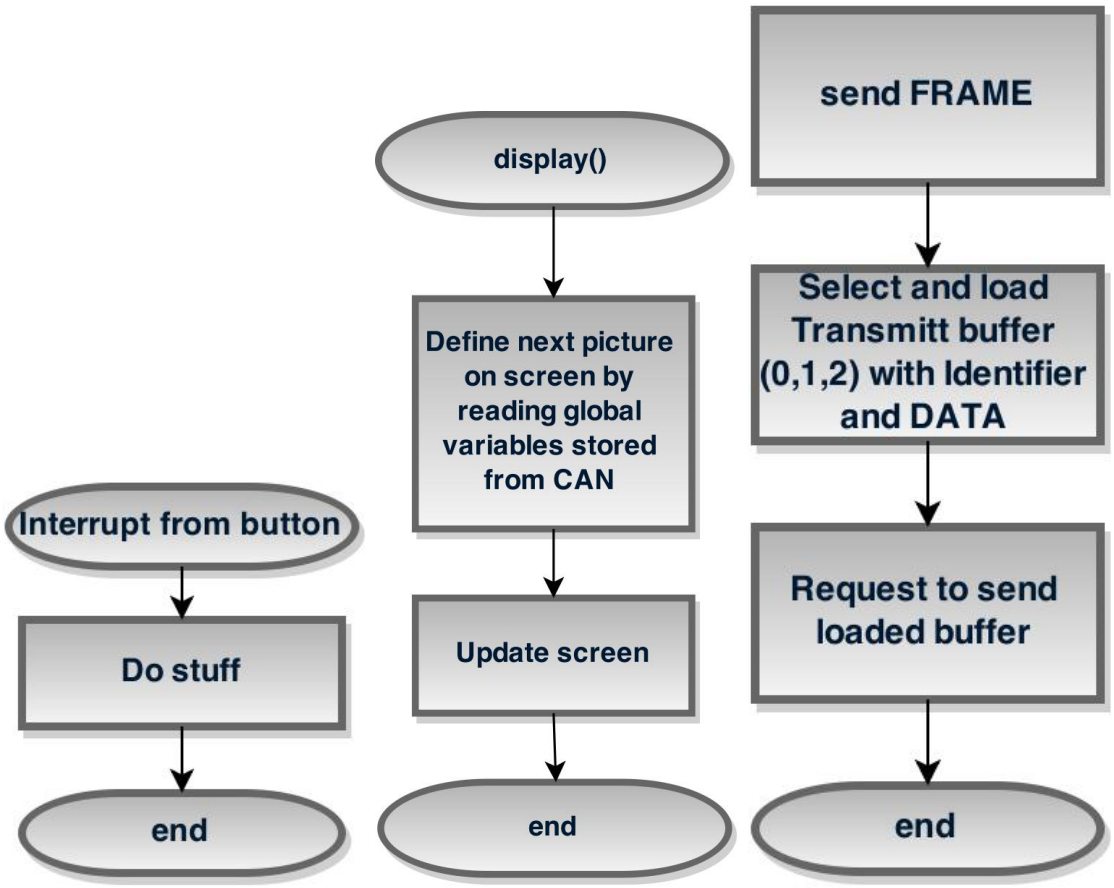


Olika temperaturen på kylvätskan i fordonet.

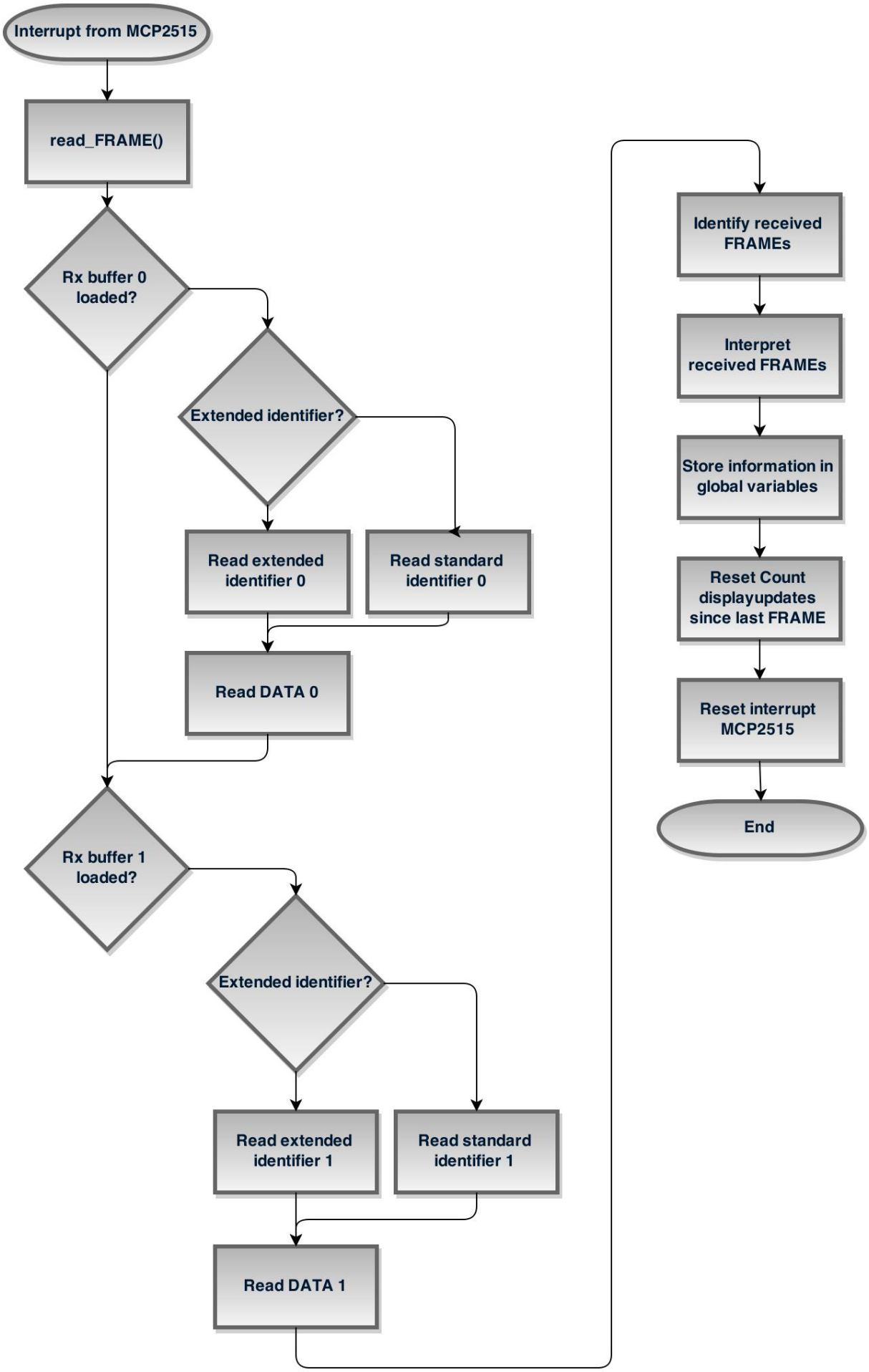


Så här ser bränslemätaren ut vid olika bränsle nivåer.

Bilaga 11 - Flödesscheman



Flödesschema för ett knapptryck, display och send Frame.



Flödesschema för att mottaga ett CAN-meddelande.