CHALMERS



Utveckling av styrsystem till förbränningsmotor för Chalmers Vera Team

Kandidatarbete inom Data- och Informationsteknik samt Tillämpad mekanik

ANTON KJELLIN
PER-SEBASTIAN PETTERSSON
FREDRIK WESTLING
LISA STENBERG

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, 2014

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Development of an engine management system for an internal combustion engine for Chalmers Vera Team

ANTON KJELLIN
PER-SEBASTIAN PETTERSSON
FREDRIK WESTLING
LISA STENBERG

© ANTON KJELLIN June 2014.
© PER-SEBASTIAN PETTERSSON June 2014.
© FREDRIK WESTLING June 2014.
© LISA STENBERG June 2014.

Examiner: ARNE LINDE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Tack till

Vi vill tacka vår handledare Anders Johansson för hans inspirerande engagemang samt Jan Björn och de andra på T-engineering för deras assistans.

Abstract

This report describes the development of an engine management system for an internal combustion engine for Chalmers Vera Team. The system is going to be used in Milage Eco-Marathon competitions which is a form of racing where the objective is to drive a car as far as possible on the energy equivalent of one litre of gasoline. Chalmers is represented by Chalmers Vera Team and the car Vera. The aim in this project was to develop and manufacture an engine management system for the engine in Vera that would give Chalmers Vera Team better abilities to handle and analyse engine data. The aim was also to give the team a good groundwork for future technical improvements that needs adaptable control. The result was a system based on two different modules named the Engine Control Unit(ECU) and the Human Machine Interface(HMI). The ECU is based on the microcontroller STM32F4 while the HMI is based on the credit-card-sized single-board computer Raspberry Pi. The ECU is controlling ignition and fuel supply to the engine based on sensor data. The HMI is handling logging of engine data, a driver interface using a screen, displaying data on the web using 3G and GPS to show the position of the car. The control algorithms for the ECU have been developed in Simulink with the add-on blockset, Waijung, from Aimagin. The microcontroller has been programmed directly from Simulink.

Sammanfattning

Den här rapporten beskriver utvecklingen av ett styrsystem till en förbränningsmotor för Chalmers Vera Team. Chalmers Vera Team tävlar i Milage Eco-Marathon som är en form av biltävling där man med en liter bensin eller motsvarande energimängd ska driva ett fordon så långt som möjligt. Chalmers deltar i tävlingen och representeras av Chalmers Vera Team och deras bil Vera. Syftet med detta projekt var att utveckla och bygga en motorstyrenhet till motorn i Vera för att ge Chalmers Vera Team bättre möjlighet att hantera och analysera motordata. Det skulle även bli en bra utvecklingsbas för att i framtiden kunna införa ny motorteknik som behöver anpassningsbar styrning. Resultatet blev ett styrsystem uppbyggt av två moduler som fick namnen ECU(Engine Control Unit) och HMI(Human Machine Interface). ECU-modulen är uppbyggd kring mikrokontrollern STM32F4 och HMI-modeulen använder enkortsdatorn Raspberry Pi med operativsystem Rasbian. ECU-modulen hanterar sensorer som mäter bland annat temperatur och lufttryck, och använder detta för att styra tändning och insprutning i motorn. HMI-modulen sköter loggning av data, presentation av data till föraren via skärm, presentation av data på internet under körning via 3G och hanterar GPS för att visa bilens position. Styralgoritmerna till ECU:n är utvecklade i Simulink. Med hjälp av ett tilläggsblockset, Waijung, från företaget Aimagin har en mikrokontroller programmerats direkt från Simulink.

Ordlista

3G	Tredje generationens mobiltelefoni.
AFR	Air-fuel ratio, massförhållande mellan luft och bensin.
AT-kommando	Styr GSM-modem och 3G/GPRS-modem.
Blockset	En samling funktionsblock till Simulink.
CPU	Central Processing Unit.
Crossplatform	Programvara som kan köras på olika plattformar.
ECU	Engine Control Unit. En av styrsystemets två moduler.
EEPROM	Icke-flyktig minnestyp.
FLASH	Transistorbaserat lagringsminne.
GNOME	Skrivbordsmiljö helt byggd på fri programvara.
GPIO	General-purpose input/output, ben på mikrokontrollern.
GPS	Global Positioning System.
GTK+	Bibliotek för att skapa grafiska gränssnitt i programmeringsspråket C.
Gtkmm	Bibliotek för att skapa grafiska gränssnitt i programmeringsspråket C++.
HMI	Human Machine Interface. En av styrsystemets två moduler.
IGBT	Transistortyp, Insulated-Gate Bipolar Transistor.
Korskompilera	Kompilera kod för en annan plattform än den man kompilerar på.
MAP	Manifold Absolut Pressure.
MOSFET	Transistortyp, fälteffekttransistor.
RPi	Raspberry Pi.
RPM	Revolutions per minute, varvtal.
Solenoid	Komponent som utför en linjär rörelse när man lägger spänning över den.
SSH	Protokoll för att ansluta sig säkert mot andra datorer över Internet.
TDC	Top dead center, kolvens position då den befinner sig högst upp i cylindern.
UART	Universal Asynchronous Receiver/Transmitter, protokoll för dataöverföring
Vera	Namnet på bilen som styrsystemet byggs till.
VNC	Virtual Network Computing, programvara för skärmdelning.
Zenerdiod	Diod med nogrann definierad backspänning.

Innehåll

1 Inledning **1**
- 1.1 Bakgrund . . . 1
- 1.2 Syfte . . . 2
- 1.3 Förutsättningar . . . 2
- 1.4 Avgränsningar . . . 3
- 1.5 Delproblem . . . 4

2 Teori **5**
- 2.1 Projektmotorns uppbyggnad . . . 5
- 2.2 Fyrtaktscykeln . . . 6
- 2.3 Förbränning . . . 7
- 2.4 Bränsleblandning . . . 8
- 2.5 Tändning . . . 9
- 2.6 Insprutning . . . 10
- 2.7 Vevaxelsensor och kamaxelsensor . . . 11
- 2.8 Bränslemappen . . . 12
- 2.9 H-brygga . . . 12
- 2.10 Icke-flyktiga minnen . . . 13

3 Metodik **15**
- 3.1 Uppdelning i två moduler, ECU och HMI . . . 15
- 3.2 Val av hårdvara till ECU . . . 16
- 3.3 Val av sensorer . . . 17
  - 3.3.1 Val av vevaxelsensor och kamaxelsensor . . . 17
  - 3.3.2 Val av temperatursensorer . . . 18
  - 3.3.3 Val av lufttrycksensor . . . 19
- 3.4 Val av drivkretsar . . . 20
  - 3.4.1 Val av drivkrets tändning . . . 20
  - 3.4.2 Val av drivkrets insprutning . . . 20
- 3.5 Utvecklingsmiljö för mjukvara till ECU-modulen . . . 20
- 3.6 Utveckling av styralgoritm till ECU:n . . . 21
  - 3.6.1 Beräkning av tid till uppgift . . . 21
  - 3.6.2 Hitta position . . . 21
  - 3.6.3 Bränslereglering . . . 22
- 3.7 Val av hårdvara till HMI . . . 22
- 3.8 Utvecklingsmiljö för HMI . . . 24
- 3.9 Val av programmeringsspråk för HMI . . . 24
- 3.10 Livedata . . . 25
- 3.11 Programmering av mappar . . . 26
- 3.12 Gränssnitt till föraren . . . 26

3.13 Testrigg 27

4 Resultat **29**
4.1 Överblick av styrsystemet 29
4.2 Simulinkmodell 30
4.2.1 Kamrutin 31
4.2.2 Vevrutin 32
4.2.3 Räknarrutin 40
4.2.4 Sensoravläsning 41
4.2.5 Periodisk hjälprutin 42
4.2.6 Loggrutin 43
4.2.7 Simulering 43
4.3 Elektronik 44
4.3.1 Spänningsförsörjning 44
4.3.2 Signalanpassning vevaxelsensor och kamaxelsensor 45
4.3.3 Signalanpassning temperatursensor 47
4.3.4 Signalanpassning MAP-sensor 48
4.3.5 Mätning av batterispänning 49
4.3.6 Knappar 50
4.3.7 Drivelektronik tändning 51
4.3.8 Drivelektronik insprutning 51
4.3.9 Kretskort 52
4.4 Programvara för HMI 55
4.4.1 Grundläggande arkitektur 55
4.4.2 Gränssnitt till föraren 56
4.4.3 Loggning 57
4.4.4 Programmering av ECU mappar 57
4.4.5 Livedata 58

5 Diskussion **61**
5.1 Genomförande 61
5.1.1 Val av programmeringsmönster 61
5.1.2 Implementation av antal varv och varvtid 61
5.1.3 Valet av Raspberry Pi och Raspbian 61
5.1.4 Valet att inte använda korskompilering 62
5.1.5 Mjukvaruutveckling i Simulink 62
5.1.6 Tankar kring det som inte täcktes in av projektet 63
5.2 Utvecklingsmöjligheter 63
5.2.1 Utökat webbgränssnitt 63
5.2.2 Hastighetsmätning 64
5.2.3 Mer sensorer och utökad info till föraren 64
5.2.4 Höghastighetsloggning 64
5.2.5 Datainsamling 64
5.2.6 Avsaknad av block för att sätta icke-periodiska mjukvaruavbrott . 65

6 Slutsats **67**

A Programmeringsmiljö för HMI **71**

B Kravspecifikation **73**

C BOSCH 0 261 230 031 **75**

1. Inledning

Rapporten beskriver ett arbete under Vera-projektet vars syfte är att få ingenjörer att växa i sin roll och ge ökade kunskaper inom hållbar utveckling. Arbetet täcker in och reflekterar över tre av Chalmers styrkeområden; Energi, Transport och Informations- och Kommunikationsteknik. Att konstruera ett eget styrsystem, för att öka möjligheterna att minska Veras energiförbrukning, har inte utförts innan och stora förhoppningar finns på att projektet ska bli en utgångspunkt för framtida utveckling.

Den globala uppvärmningen anses vara ett av våra största problem i dagens samhälle och forskning riktad mot minskad energiförbrukning är en viktig del för att vända utvecklingen. Arbetet, tillsammans med övriga Vera-projekt, ska förhoppningsvis ge både studenter och Chalmers nya insikter, kunskap och tankesätt för att driva fram nya innovationer inom ämnet.

1.1 Bakgrund

Milage Eco-Marathon är en tävlingsform där man med en liter bensin eller motsvarande energimängd ska driva ett fordon med förare så långt som möjligt, utmaningen är att bygga den energisnålaste bilen. Chalmers deltar i klassen Bensindrivet och representeras av Chalmers Vera Team med bilen Vera(se figur 1.1). Den största tävlingen hålls i Rotterdam och där deltar Chalmers Vera Team tillsammans med drygt sextio andra lag från hela Europa och har placerat sig på en niondeplats som bäst. I Rotterdam måste bilen ta sig 16.117 km på under 39 minuter. Efter avklarat race mäts hur mycket bensin som förbrukats. Sedan beräknas den sträcka bilen skulle kunna köra på en liter bensin.

Teamet består av studenter från olika program på Chalmers som tillsammans varje år hjälps åt att vidareutveckla Vera där stort fokus läggs på motorn. Teamet väljer varje år ut olika delar som skall utvecklas vidare, vissa delar utvecklas av teamet själva, medan andra utvecklas i kandidatarbeten och exjobb.

Motorn till Vera är en av många delar som är utvecklad och byggd av studenter på Chalmers. Däremot används idag ett kommersiellt styrsystem. Ett styrsystem är den dator som styr motorns tänd- och bränslesystem. Detta görs med hjälp av ett antal sensorer som sitter kopplade till motorn samt inställningsbara parametervärden.



Figur 1.1: Chalmers Vera Team och Verabilen [1]

Dagens styrsystem begärnsar möjligheterna att analysera motordata då teamet bara kan avläsa data från motorn med en dator direkt inkopplad, alltså endast när Vera körs i testbänk. För att ytterligare kunna förbättra energiförbrukningen och för att underlätta utveckling av motorn vill teamet kunna logga motordata.

Förra året gjordes ett kandidatarbete där man kopplade in sig på det befintliga styrsystemet[1]. Med hjälp av Bluetooth och en Android-telefon försökte man sedan logga motordata samt skicka det till en webbserver. Baserat på deras erfarenheter bestämde sig teamet i år för att bygga ett helt nytt styrsystem.

1.2 Syfte

Syftet med projektet är att utveckla och konstruera ett styrsystem som skall ge Chalmers Vera Team bättre möjlighet att hantera och analysera motordata. Det skall även ge Chalmers Vera Team en bra utvecklingsbas för att i framtiden kunna införa ny motorteknik som behöver anpassningsbar styrning.

1.3 Förutsättningar

Chalmers Vera Team hade innan projektets start tagit fram en lista med krav på vad styrsystemet skulle ha för funktioner, utformning och kapacitet. Listan gjordes om till en tydligare kravspecifikation, se bilaga Kravspecifikation. Ytterligare ett önskemål från

[1] Bildkälla: `http://chalmersverateam.se/portfolio/`

Chalmers Vera Team var att styralgoritmerna för styrsystemet skulle utvecklas i Simulink.

Styrsystemet skulle utvecklas för Veras tävlingsmotor(Figur 1.2) som är en encylindrig portinsprutad fyrtaktsmotor med överliggande kamaxel och två ventiler. Motorn har inget gasspjäll utan går alltid på fullast. Den har en slagvolym på 47,5 cm^3, ger en effekt på 2,5 hk och är konstruerad och byggd av studenter på Chalmers. En mer ingående beskrivning av Veras motor återfinns i kapitel 2.

1.4 Avgränsningar

Att implementera styrsystemet i Verabilen avgränsades från projektet. Istället användes en exprimentmotor, Honda GX35(Figur 1.3), för att verifiera styrsystemet. Exprimentmotorn har liknande storlek och utförande som motorn i Vera. Den är dock inte portinsprutad i sitt orginalutförande men delar för att konvertera den har tillhandahållits av Chalmers Vera Team. Styrsystemet har heller inte byggts in i en vattentät låda som nämns i Kravspecifikationen.



Figur 1.2: Motorn(mk6) som nu används i Vera



Figur 1.3: Experimentmotor som använts under projektets gång - Honda GX35

På grund av den begränsade tiden uteslöts ett antal krav från kravspecifikationen. Styrsystemet kommer inte:

- analysera motorns acceleration och meddela att den följer, eller inte följer, en förväntad acceleration.
- larma och ge felkoder vid fel.
- göra kompensationer för att undvika knack.
- kompensera för förändringar i luftfuktigheten.

Fokus i projektet har legat på att få en fungerande bas som är förbered för utbyggnad. Dessa krav beskriver funktionallitet som inte är nödvändig för styrsystemets grundfunktion.

1.5 Delproblem

Ett antal delproblem för uppfylla projektets syfte identifierades och följande uppdelning gjordes.

Hårdvara Systemet behöver hämta in data utifrån, göra beräkningar, spara och presentera data med mera. Detta innebär att mycket hårdvara behöver införskaffas, att välja hårdvara som är välanpassad för ändamålet är en del av projektet.

Styralgoritm För att styra motorn behöver en styralgoritm implementeras. Denna algoritm styr tändning och insprutning av bränsle beroende på ett antal insignaler från sensorer och inställningsbara parametervärden. Som nämnts under avsnitt 1.3 är ett önskemål från Chalmers Vera Team att styralgoritmen utvecklas i Simulink.

Signalhantering och mätning För att kunna förse styralgoritmen med indata används sensorer. Dessa sensorer kan inte kopplas in direkt till mikrokontrollen utan behöver kretsar som anpassar signalen. Dessa kretsar bör även skydda mot användarfel, störningar och kortslutningar.

Drivelektronik för aktuatorer För att kunna styra aktuatorerna ifrån mikrokontrollen behövs drivelektronik eftersom tillräckligt stora strömmar och spänningar inte kan ges direkt från mikrokontrollen.

Loggning av data För att möjligöra analyser av motordata behöver data lagras och göras tillgänglig efter ett race. Då många parametrar skall loggas kommer stora mängder data genereras. Detta innebär att lagringen måste ske på ett effektivt och organiserat sätt för att så mycket relevant information som möjligt skall kunna sparas och göras tillgänglig.

Gränssnitt med föraren Olika typer av live- och aggregerad data kan ge föraren bättre uppfattning av hur racet fortlöper. För att kunna ge föraren bättre förutsättningar att utföra ett bra race så skall ett grafiskt gränssnitt som visar relevant information tas fram.

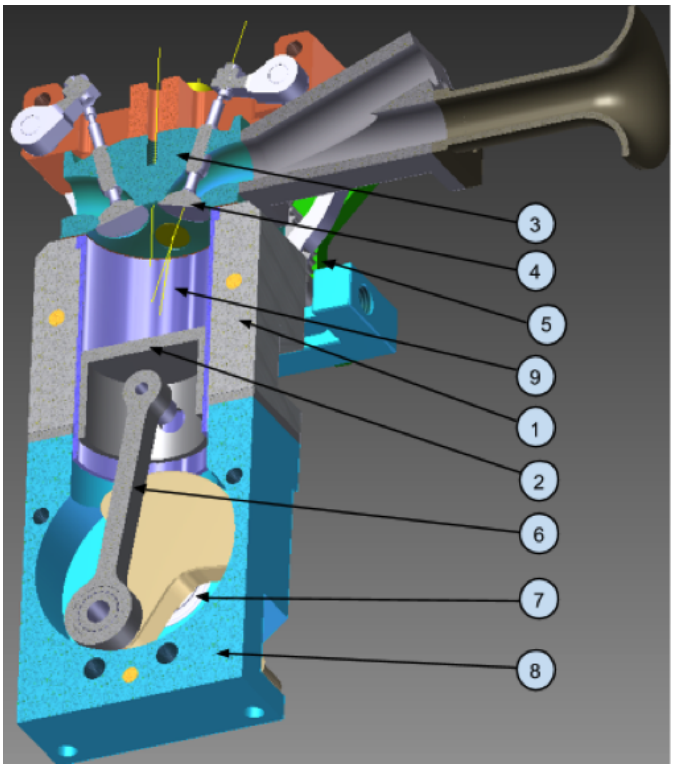
Presentation av livedata För att förbättra möjligheterna för teamet att under race analysera hur motorn presterar skall motordata skickas till en webbserver. Detta möjliggör att man via internet kan följa hur motorn beter sig.

2. Teori

I det här kapitlet förklaras de teoribegrepp som är fundamentala för resten av rapporten. Kapitlet täcker grundläggande teori om hur en fyrtaktsmotor fungerar med fokus på de delar och faktorer som är viktiga för styrsystemets funktion samt annan teori som krävs för att förstå vissa av de lösningar som senare presenteras.

2.1 Projektmotorns uppbyggnad

Motorn projektet riktats mot är en encylindrig portinsprutad bensinmotor av fyrtaktstyp med överliggande kamaxel och två ventiler. Förutom tändsystem och bränslesystem består motorn av ett antal huvudkomponenter; cylinder, kolv, topplock, ventiler, kamaxel, vevstake, vevaxel och vevhus. Figur 2.1 visar en veramotor i genomskärning. Flera ingående komponenter är synliga i bilden.



Figur 2.1: Veramotorns uppbyggnad; cylinder(1), kolv(2), topplock(3), ventiler(4), kamaxel(5, dold), vevstake(6), vevaxel(7, dold) och vevhus(8), förbränningsrum(9)

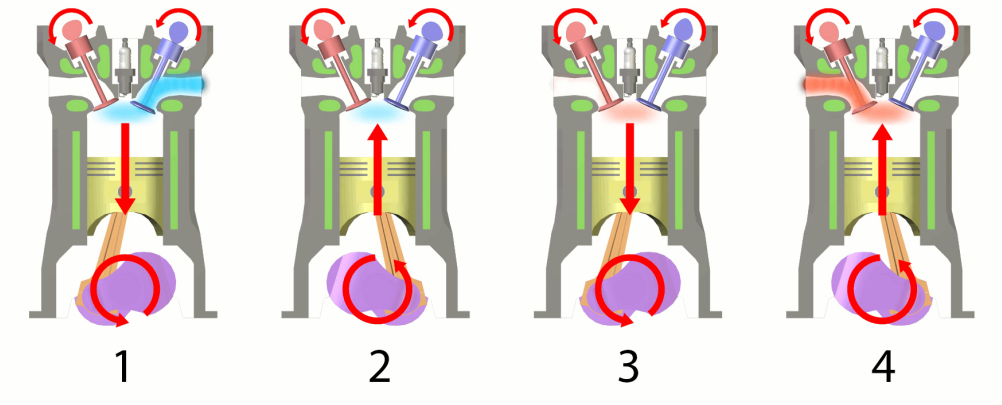
Kemisk energi i bränslet omvandlas till mekanisk energi då förbränning sker i förbränningsrummet. Förbränningsrummet är den volym som avgränsas av cylinderväggen, topplocket och kolvens ovansida. Kolven sitter i cylindern och utför en linjär rörelse

från övre vändläge neråt till nedre vändläge där den vänder och fortsätter uppåt. Vevstaken kopplar kolven mekaniskt till vevaxeln som sitter lagrad i vevhuset. Vevstaken överför kolvens linjära rörelse till rotation på vevaxeln. När motorn är i drift omvandlas kraft på kolven till vridmoment på vevaxeln.

Ventilerna är monterade i topplocket och tillåter via kanaler åtkomst till förbränningsrummet utifrån. Via insugsventilen tillförs bränsleblandning till förbränningen och via avgasventilen avlägsnas avgaserna från förbränningen. Ventilerna styrs av kamaxeln som också är monterad på topplocket. Kamaxeln roterar med halva vevaxelns hastighet och öppnar ventilerna då den befinner sig i en bestämd position. Denna position motsvaras av en position hos kolven då kamaxelns rotation är knuten till vevaxelns via en kamrem.

2.2 Fyrtaktscykeln

Fyrtaktscykeln består av fyra efter varandra följande så kallade takter; insugs-, kompressions-, arbets- och avgastakten, vardera på 180 vevaxelgrader. Motorns totala period är 720 grader, kolven rör sig under denna upp och ned två gånger.



Figur 2.2: Fyrtaktscykelns fyra takter: insugs(1)-, kompressions(2)-, arbets(3)- och avgastakten(4) [1]

Under insugstakten, figur 2.2(1), rör sig kolven från övre vändläget neråt i cylindern till sitt nedre vändläge, insugsventilen är då öppen och bränsleblandning sugs in i det expanderande förbränningsrummet.

Efter insugstakten följer kompressionstakten, 2.2(2). Ventilerna är stängda och kolven rör sig uppåt i cylindern och komprimerar bränsleblandningen, trycket och temperaturen ökar i förbränningsrummet. När kolven åter närmar sig övre vändläge antänds bränsleblandningen och börjar förbrännas.

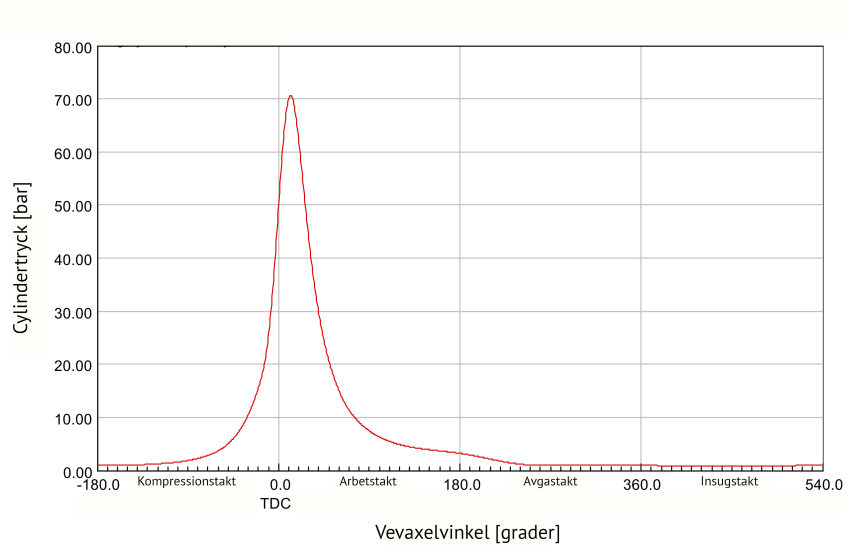
[1] Bildkälla: `http://en.wikipedia.org/wiki/Four-stroke_engine`

Då övre vändläge efter insugstakten passeras inleds arbetstakten, 2.2(3), bränsleblandningen förbränns, värme avges och trycket i förbränningsrummet ökar ytterligare. Detta tryck skapar en kraft på kolven och trycker den neråt under arbetstakten, moment överförs till vevaxeln. Då kolven under arbetstakten rör sig neråt expanderar förbränningsrummet och trycket minskar. När kolven åter är i nedre vändläge öppnas avgasventilen, kolven börjar röra sig uppåt igen och avgastakten inleds.

Kolvens rörelse uppåt under avgastakten, 2.2(4), trycker ut avgaserna ur förbränningsrummet, genom avgasventilen och ut i avgassystemet. När kolven efter avgastakten når övre vändläge öppnas åter insugsventilen och en ny arbetscykel inleds.

2.3 Förbränning

Då gnistan slår börjar bränsleblandningen brinna och en flamfront sprider sig från tändstiftet ut genom bränsleblandningen som successivt förbränns. Förbränningen frigör energi och gasen i förbränningsutrymmet värms upp, vilket i enighet med ideella gaslagen skapar en tryckökning i förbränningsutrymmet. Tryckökningen ger en kraft på förbänningrummets väggar och kraften på kolvens ovansida trycker kolven nedåt i cylindern. När kolven trycks nedåt ökar volymen i förbränningsrummet, vilket verkar för att minska trycket. Det totala trycket i förbränningskammaren varierar med vevaxelvinkeln, och vid vilken vinkel maximala trycket hamnar beror på när bränsleblandningen antänds.



Figur 2.3: Exempelgraf, vevaxelgrader-cylindertryck

Arbete plockas ut från motorn på vevaxeln. Storleken på vevaxelmomentet, som kraften på kolven ger upphov till, beror på vevstakens vinkel. Vevstaken förbinder kolven med vevaxeln och deras position relativt varann bestäms av motorns geometri. Vid övre-

och nedre vändläge är kraftöverföringen noll, däremellan är den större än noll och det finns vevaxelvinklar under arbetstakten vid vilka det är mer fördelaktigt att ha ett högt tryck i förbränningskammaren om målet är att få ut så stort moment som möjligt från förbränningen. Totala effektiva arbetet under arbetsfasen varierar också beroende på när antändning sker och det finns en optimal vinkel för antändning. Man strävar alltså efter att hela tiden tända så att maximala trycket, trycktoppen, i förbränningsutrymmet inträffar vid en viss vevaxelvinkel som ges av motorns geometri. Figur 2.3 visar en tryckkurva(trycket i förbränningsrummet under en arbetscykel). Trycktoppen återfinns i figuren strax efter att kolven har passerat TDC.

Då tryckminskningen till följd av kolvens rörelse, till skillnad från förbränningens hastighet, är beroende av kolvens hastighet kommer denna optimala tändvinkel variera med motorns varvtal. Det är därför viktigt att med styrsystemet kunna ställa in vid vilken vevaxelvinkel antändning ska ske vid olika varvtal.

Knack
Knack uppstår då bränsleblandningen antänds av en annan källa än tändstiftet. Denna oavsiktliga antändning kan bero på olika faktorer så som höga temperaturer och höga tryck. Knack ger upphov till tryckspikar i förbränningskammaren som kan vara skadliga för motorn. Detta leder också till vibrationer av en för varje motor bestämd frekvens. Vibrationerna kan utnyttjas för att detektera knack innan det blir skadligt för motorn. Knack kan undvikas genom att köra en fetare bränsleblandning eller att senarelägga tändningen. Vid senareläggning av tändningen flyttas trycktoppen i förbränningsrummet framåt, längre från TDC. I figur 2.3 motsvarar detta en förflyttning av kurvan till höger.

2.4 Bränsleblandning

Bränsleblandningen består av luft och bensin. Förhållandet mellan massan luft och massan bensin i blandningen betecknas AFR(ekvation 2.1) och är mycket viktigt för förbränningen.

$$AFR = \frac{m_{luft}}{m_{bensin}} \tag{2.1}$$

Den blandning där precis tillräckligt mycket luft finns tillgänglig för att all bensin kan förbrännas kallas för stökiometrisk blandning och betecknas AFR_{stok}. Förbränningens lambda-värde(ekvation 2.2) uttrycker hur den förbrända blandningen förhåller sig till bränslets AFR_{stok}.

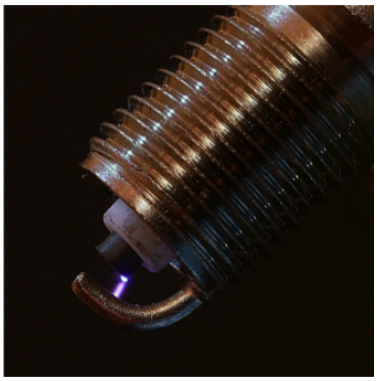
$$\lambda = \frac{AFR}{AFR_{stok}} \tag{2.2}$$

Bensin har ett AFR_{stok} som ligger kring 14,7. Men eftersom den kemiska sammansättningen skiljer sig från bensin till bensin kan också AFR_{stok} skilja sig något. Stökiometriska blandningar ger alltid ett lambda på 1(ekvation 2.2).

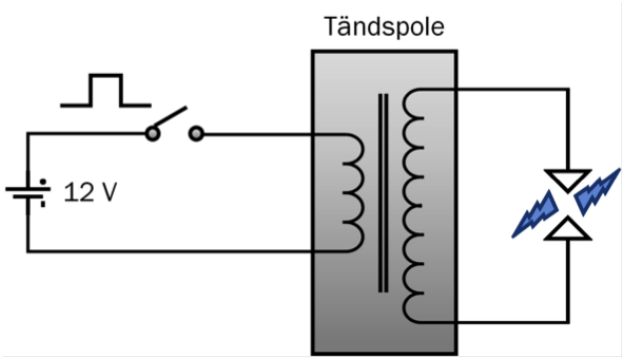
Blandningar med för mycket bensin kallas feta och får ett lambda mindre än 1, blandningar med för lite bensin kallas magra, och får ett lambda större än 1. Blandningar långt ifrån $\lambda = 1$ brinner inte, endast inom ett intervall brinner bränsleblandningen och arbete kan plockas ut från motorn. En viss blandning kommer att ge ett största möjliga arbete ut per tillförd massa bensin. Det är därför viktigt att med styrsystemet kunna styra förhållandet mellan bränsle- och luftmassa.

2.5 Tändning

Gnistan som startar förbränningen bildas mellan två elektroder i tändstiftet(Figur 2.4). Gnistan skapas när spänningen mellan elektroderna blir tillräckligt hög(ca 40-100 kV[2]) så gasen som omsluter elektroderna joniseras och blir ledande. Den stora spänningen skapas med hjälp av en tändspole. Tändspolen är en konstruktion av två spolar, en primärspole lindad runt en sekundärspole Sekundärspolen har många fler lindningar än primärspolen. Figur 2.5 visar en principskiss över en tändspole.



Figur 2.4: Gnista bildas mellan tändstiftets två elektroder[2]



Figur 2.5: Illustration av tändspolens funktion

När brytaren sluts går en ström i primärspolen som ger upphov till ett magnetfält. Detta magnetfält inducerar en spänning i sekundärspolen och energi laddas upp i tändspolen. När brytaren öppnas igen försvinner magnetfältet från primärspolen. Den snabba förändringen i magnetfält ger upphov till en spänningsspik i sekundärspolen och en gnista bildas[3].

[2]Bildkälla: `http://sv.wikipedia.org/` Sökord: Tändstift

Tiden som brytaren är sluten brukar kallas laddtid(eng.dwell time) och ligger, för tändspolar som används i Vera, runt 3 ms. Om laddtiden är för stor kan tändspolen förstöras och om den är för liten bildas ingen gnista[4]. Batterispänningen påverkar laddtiden, med en lägre batterispänning behöver laddtiden ökas.

Brytaren, i bilden(Figur 2.5), är ofta en transistor som styrs av styrsystemet och kallas drivkrets.

2.6 Insprutning

Tillförseln av bränsle görs av spridaren(Figur 2.6). En spridare är en elektriskt styrd ventil som, när den öppnar, tillåter trycksatt bränsle att passera via ett munstycke. Spridaren är uppbyggd av en solenoid med en nål som järnkärna. Nålen hindrar bränslet från att passera spridaren. När solenoiden spänningssätts går en ström genom spolen och ett magnetfält skapas. Magnetfältet trycker upp nålen och låter bränsle passera.

För att stänga spridaren används traditionellt en fjäder, spänningen bryts och nålen trycks tillbaka av fjädern. I vissa modernare styrsystem vänds istället polariteten över solenoiden, nålen trycks då tillbaka av det omvända magnetfält som skapas.

Spridare brukar delas upp i två kategorier, lågimpediva($1 - 5\Omega$) och högimpediva($12 - 16\Omega$)[5]. Lågimpediva spridare behöver en strömbegränsande styrning för att inte gå sönder. Spridaren som används i projektet är högimpediv och öppnas då den matas dem med 12 V.



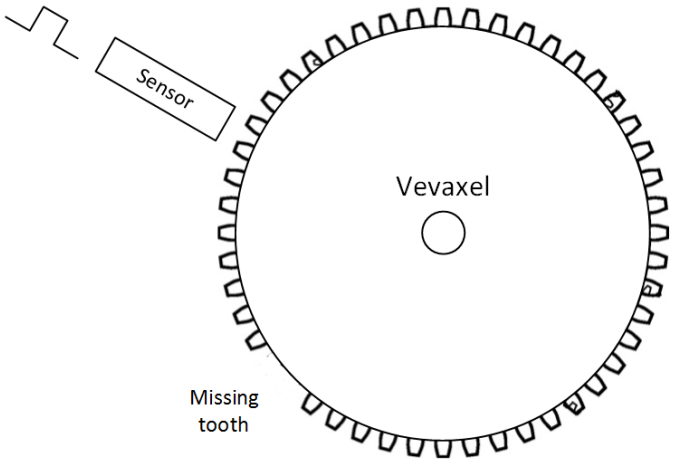
Figur 2.6: Spridaren som används i motorn(16450-GEV-761). Nedre sidan ansluts mot motorn, längst upp anluts slang med trycksatt bränsle. I bilden syns även kontakten för strömanslutning.

Förhållandet mellan hur länge spridaren är öppen och mängden bränsle som sprutas in är, i användningsområdet, linjärt. Detta förhållande beskrivs av en spridarkonstant(eng. flowrate) som påverkas av bränslets tryck.

En tumregel är att spridaren under motorns arbetscykel skall stängas precis innan insugsventilen öppnas.

2.7 Vevaxelsensor och kamaxelsensor

För att styrsystemet skall veta motors position i arbetscykeln används vanligtvis två sensorer, en sensor på vevaxeln och en på kamaxeln som båda mäter diskreta avkänningspunkter. Vevaxelsensorn har flera avkänningspunkter på ett svänghjul som roterar med vevaxeln. På svänghjulet finns ofta en lucka där det saknas en eller fler avkänningspunkter(eng. missing tooth). Kamaxelsensorn har endast en avkänningspunkt som är placerad på kamaxeln. Kamaxeln roterar med halva vevaxelns hastighet.



Figur 2.7: En vevaxelgivare som känner av kuggarna på hjulet. I nedre delen av bilden syns missing tooth.

Styrsystemet använder sensorn på vevaxeln(Figur 2.7) för att kontinuerligt uppdatera motorns position. Sensorn på kamaxeln samt missing tooth används som referenspunkter i motorns arbetscykeln. Referenspunkten på kamaxeln behövs eftersom vevaxeln roterar två varv per arbetscykel. Med endast missing tooth kan styrsystemet ej avgöra vilken halvcykel som motorn befinner sig i. Att styrsystemet vet motorns position är helt avgörande för dess funktion.

2.8 Bränslemappen

Motorns varvtal och gasspjällets vinkel(lasten) avgör hur stor volym luft som sugs in i cylindern under insugstakten. Detta värde kan anges som en andel av totala cylindervolymen, fyllnadsgraden. En vanlig metod för styrsystem är att beskriva förhållandet mellan varvtal, last och fyllnadsgrad med en så kallad bränslemapp(Figur 2.8). Detta är en tabell i vilken fyllnadsgraden vid ett antal varvtal och laster kan slås upp. Denna mapp utgör en inställningsmöjlighet i styrsystemet och ställs in vid mappning av motorn. Mappen används av styrsystemet vid beräkning av hur mycket bränsle som skall tillföras förbränningen under en arbetscykel.

Last %

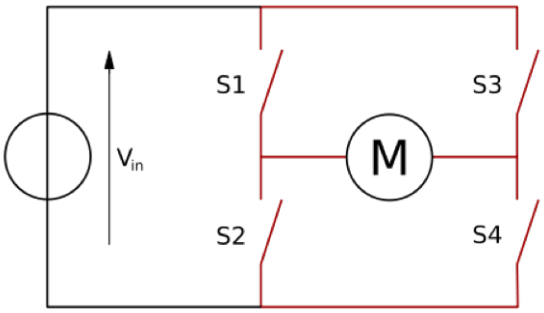
RPM	21,4	21,9	22,3	22,7	23,1	23,4	23,8	24,2	25,1	25,8	26,6	27,4	28,2
800	34	34	35	48	48	61	64	70	76	81	86	90	94
1000	34	34	35	48	48	61	64	70	76	81	86	90	94
1200	32	32	32	38	38	58	61	66	72	76	83	88	95
1400	30	30	30	33	33	55	59	64	68	72	76	82	90
1600	28	28	28	29	29	52	55	59	63	68	72	76	88
1800	27	27	27	28	28	49	53	57	61	67	71	75	89
2000	24	24	24	26	26	45	50	55	59	64	69	74	87
2200	23	23	23	23	23	45	49	53	57	59	64	70	81
2400	22	22	22	22	22	38	43	48	53	56	59	63	74
2600	22	22	22	22	22	38	41	46	50	52	55	59	70
2800	22	22	22	21	21	34	39	46	50	54	56	60	71
3000	22	22	22	21	21	33	37	45	50	51	55	59	71
3200	21	21	21	21	21	29	36	45	50	53	55	60	74
3400	21	21	21	22	22	28	34	45	50	53	55	60	74
3600	21	21	21	22	22	29	31	43	50	54	57	60	74
3800	21	21	21	23	23	28	30	43	50	55	58	62	74
4000	21	21	21	23	23	28	32	41	50	57	61	64	75
4200	12	12	12	22	22	30	32	41	50	58	62	64	74
4400	22	22	22	21	21	31	32	41	51	57	62	64	73
4600	22	22	22	20	20	28	31	40	52	55	59	62	71
4800	22	22	22	20	20	25	30	38	52	50	56	58	73

Figur 2.8: Exempel på en bränslemapp

2.9 H-brygga

H-bryggan är en elektisk krets som gör att polariteten kan ändras över en elektrisk last, exempelvis en spridare. H-bryggor används ofta vid styrning av DC-motorer för att kunna ändra motorns rotationsriktning samt bromsa motorn.
En illustration av H-bryggans konstruktion kan ses i Figur 2.9. Då S1 och S4 sluts(S2 och S3 öppna) blir polariteten över lasten, M, positiv. Då S2 och S3 sluts(S1 och S4 öppna) blir polariteten istället negativ. Öppnas alla brytare så ligger ingen spänning över lasten. H-bryggor kan köpas som integrerade kretsar men kan även byggas av exempelvis MOSFET-transistorer.

[3]Bildkälla: http://sv.wikipedia.org/wiki/H-brygga



Figur 2.9: Enkel illustration av en H-brygga[3]

2.10 Icke-flyktiga minnen

För att kunna lagra data permanent i en mikrokontroller, så att de finns sparade efter omstart, behövs icke-flyktigt minne. Icke-flyktigt minne är en typ av minne som behåller den lagrade datan även när minnet inte får någon ström. I persondatorer används oftast hårddiskar och minneskort. Många mikrokontrollers har ett inbyggt FLASH- eller EEPROM-minne, dessa minnen är icke-flyktiga.

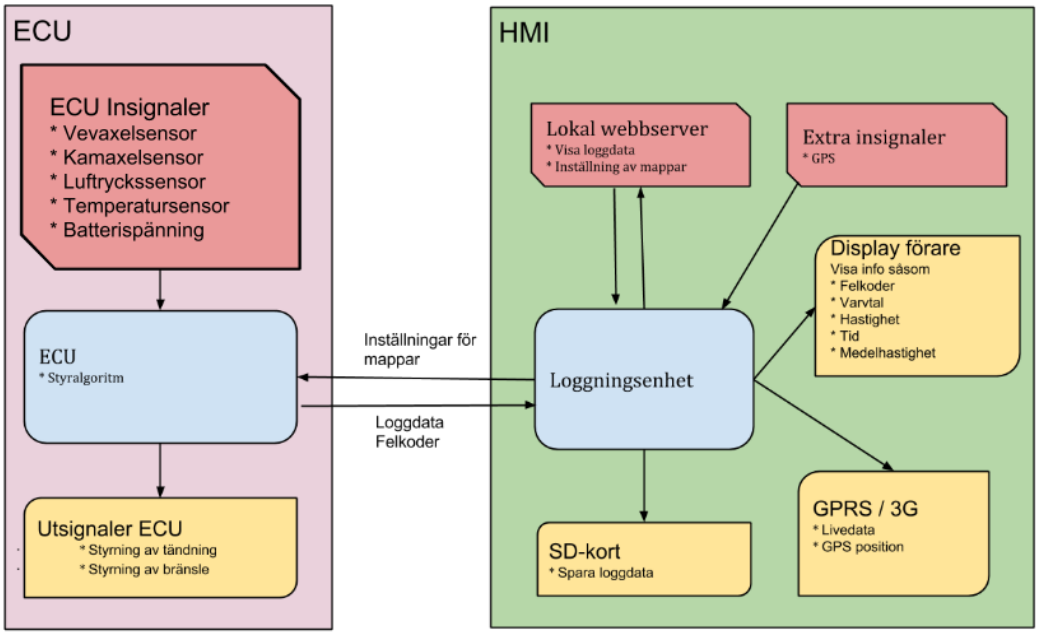
3. Metodik

Här presenteras genomförandet av projektet, de val och de resonemang som har lett fram till resultatet.

3.1 Uppdelning i två moduler, ECU och HMI

Efter en grundläggande analys av kravspecifikationen konstaterades att kraven kunde delas in i två kategorier. Delar av kraven riktas mot styrningen av motorn, andra mot hantering och presentation av loggdata. De två kategorierna ställde skillda krav på hårdvara och mjukvara, därför delades styrsystemet upp i två olika moduler. Detta innebar att hårdvaran och mjukvaran bättre kunde anpassas efter kraven. Modulerna fick benämningarna ECU(Engine Control Unit) och HMI(Human Machine Interface).

ECU-modulen hanterar sensorerna samt styr tändning och bränsletillförsel. HMI-modulen hanterar loggning av data, gränssnitt till förare, GPS-positionering, mätning av hastighet, uppladdning av data till internet etc. Uppdelningen mellan ECU och HMI illustreras i Figur 3.1.



Figur 3.1: Uppdelning av styrsystemet

3.2 Val av hårdvara till ECU

Till ECU-modulen eftersöktes en mikrokontroller på ett utvecklingskort, detta för att snabbt kunna komma igång med utvecklingen. Då styrsystemet skall hantera många analoga sensorer var det viktigt att mikrokontrollern hade tillräckligt med AD-omvandlare. Som framgår i kravspecifikationen behövdes plats för åtta analoga sensorer och för att förbereda styrsystemet för ny teknik önskades plats för fler.

Andra egenskaper som eftersöktes var att mikrokontrollern skulle tillåta möjlighet att kommunicera med HMI-modulen, vara väldokumenterad samt kunna programmeras med Simulink.

Valet föll på mikrokontrollern STM32F4 från STMicroelectronics. Det finns många utvecklingskort på marknaden för STM32F4 men STM32F4-Discovery(Figur 3.2) valdes till projektet då den tillverkas av STMicroelectronics själva och en av de vanligaste på marknaden.



Figur 3.2: STM32F4 Discovery, utgör kärnan i ECU-modulen

STM34F4 har en 32-bitars ARM-processor med en klockfrekvens på 168 MHz. Den har även 16 st 12-bitars AD-omvandlare samt 5V-toleranta I/O-portar.

Kortet har inbyggd programmerare och kan programmeras direkt via inbyggd USB-port. Mikrokontrollern drivs med 3,3 V men kortet har en inbyggd spänningsregulator så extern 5V-källa eller USB-kabel kan användas. På kortet finns också två knappar och fyra lysdioder som kan användas vid testning och experimenterande. Dessa kom ofta till an-

vändning vid felsökning.

AD-omvandlarna i mikrokontrollern mäter spänningar mellan 0 V och 3.3 V.

3.3 Val av sensorer

Här redovisas resonemangen kring sensorvalen.

3.3.1 Val av vevaxelsensor och kamaxelsensor

De för ändamålet två vanligaste typerna av sensorer, VR-sensorer(Variable Reluctance) och halleffektsensorer, undersöktes till projektet.

VR-sensorn är den klassiska sensorn till personbilar. Det är en passiv komponent som består av en spole med en järnkärna. Exempel på VR-sensors funktion kan ses i Figur 2.7. När tänderna på hjulet snurrar förbi induceras en spänning i spolen. Signalen blir sinusformad med en amplitud som ökar med varvtalet. Med hjälp av en krets anpassad för ändamålet kan en digital signal, som enkelt kan hanteras av en mikrokontroller, erhållas. Exempel på sådana kretsar är NCV1124 och LM1815.

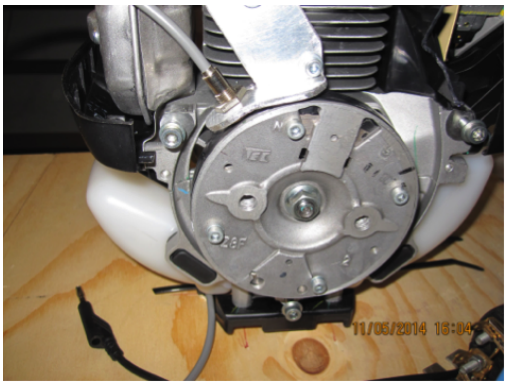
Halleffektsensorn är en aktiv komponent som ger en digital utsignal. Sensorn känner av magnetfält och beroende på om ett magnetfält detekteras eller inte ges antingen hög eller låg signal. Sensorn monteras som i Figur 3.3 samt Figur 3.4).

Fördelar med halleffektsensorn[6]:

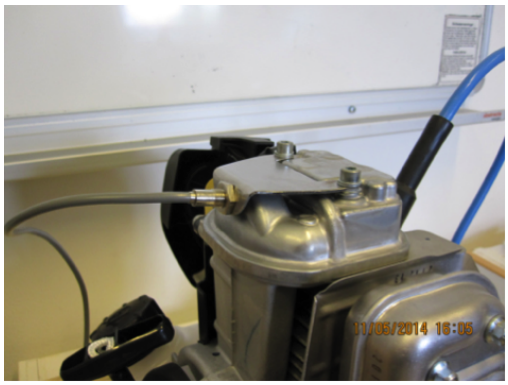
- Robust mot elektromagnetiska störningar.
- Klarar att avläsa låga hastigheter.
- Mindre känslig för hur den monteras då den klarar större avstånd (upp till 5 mm) samt har högre tolerans för vilken vinkel den kan ha mot mätobjektet.

Fördelarna med VR-sensorn är att den klarar högre temperaturer(över 300 $^{\circ}C$) samt att den är billigare[7].

Halleffektsensorn valdes till projektet pågrund av att den är mindre känslig för störningar och hur den monteras samt att den ger en digital signal utan extern krets. Det är även en lösning som är bättre anpassad för framtiden då Chalmers Vera Team skall använda sig av denna teknik i en ny motor som är under utveckling. Den specifika sensor som valdes är en likadan som Chalmers Vera Team kommer använda i framtiden, “Hall Effect Proximity Switch CYKN8-02CL0. Denna sensor behöver ett motstånd mellan signal och 5 V(eng. pull-up resistor) på 1-10 $k\Omega$ för att fungera.



Figur 3.3: Halleffektsensor monterat vid pulshjul på testmotorns vevaxel



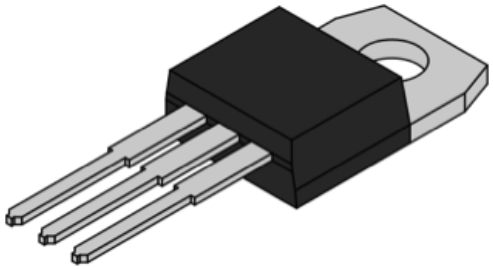
Figur 3.4: Halleffektsensor monterad vid testmotors kamaxel

3.3.2 Val av temperatursensorer

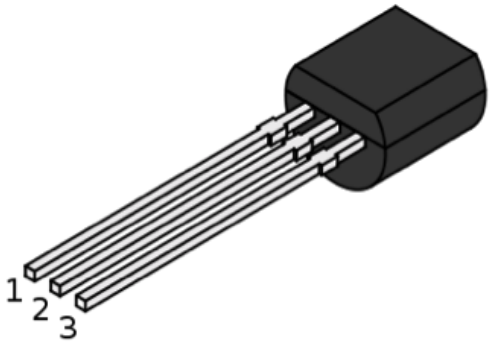
Från kravspecifikationen framgår att fyra temperatursensorer behövs. Tre som mäter temperaturen på motorn och en som mäter lufttemperaturen. För att spara tid togs beslutet att använda samma typ av sensor på samtliga platser. Sensorerna till motorn behövde enligt Chalmers Vera Team klara av temperaturer upp till 150 $°C$. De behövde även på ett smidigt sätt kunna fästas i motorn.

Första förslaget som övervägdes var att använda sig av NTC-motstånd(termistor). Det är en komponent vars elektriska motstånd varierar med temperaturen. NTC-motstånd har fördelarna att de är små, passiva och billiga komponenter. Nackdelarna är att de inte är linjära, kräver ganska omfattande elektronik för att fungera samt att de måste kalibreras[8]. Därför togs beslutet att använda aktiva, linjära kretsar.

Sensorn som valdes är LM35. Sensorn spänningsätts med 5 V och ger sedan en utsignal på 10 $mv/°C$. Den klarar att mäta temperaturer från -10 $°C$ till 150 $°C$. LM35 tillverkas i olika kapseltyper. Kapseltypen TO-220(Figur 3.5) valdes för de som skall sitta på motorn eftersom de kan skruvas fast genom hålet i flänsen. För lufttemperatursensorn valdes formatet TO-92(Figur 3.6), sensorn kan monteras i en kabelsko och sedan sättas på valfri plats i bilen.



Figur 3.5: .Kapseltyp TO-220[1]



Figur 3.6: Kapseltyp TO-92[2]

3.3.3 Val av lufttrycksensor

Luftryckssensorer till styrsystem kallas MAP-sensorer(Manifold Absolute Pressure = Insugningsrörets Absoluta Tryck) och mäter trycket i insuget. Det finns grovt sett två typer av MAP-sensorer som är anpassade efter olika tryck, ena typen för undertryck, andra för övertryck. På Vera används inte turbo så det kommer inte vara övertryck i insuget. Det kommer heller inte vara stort undertryck då Vera inte stryper lufttillförseln utan går alltid på fulllast. Sensorn behöver därför endast mäta normala tryckförändringar som beror på vädret. Mätområde skall därför ligga runt 1 bar.

Sensorn som valdes är BOSCH 0 261 230 031(Figur 3.7). Den är designad för små personbilar och är anpassad efter standardtryck(atmosfärstryck).



Figur 3.7: MAP-sensor(lufttryckssensor), BOSCH 0 261 230 031

[2]Bildkälla `http://en.wikipedia.org/wiki/TO-220`
[2]Bildkälla `http://en.wikipedia.org/wiki/TO-92`

3.4 Val av drivkretsar

Här redovisas utvecklingen av drivkretsarna och resonemangen kring valen av komponenter.

3.4.1 Val av drivkrets tändning

Drivkretsar byggda för tändning finns i många olika utföranden. Kretsarna VB921 och BIP373 har Chalmers Vera Team tidigare använt sig av. I konventionella styrsystem används ofta BIP373(Bosch integrated power) och liknande kretsar. I modernare styrsystem används ofta IGBT-baserade kretsar, eftersom dessa är snabbare och mer effektiva[9]. Av denna anledning togs också beslutet att välja en modernare IGBT-krets till projektsystemet.

3.4.2 Val av drivkrets insprutning

Chalmers Vera Team vill kunna stänga spridaren genom att vända på polariteten, se krav 4 i Kravspecifikation. Först designades dock en testkrets där spridaren stängdes på traditionellt vis, med fjäder, detta för att göra utvecklingen enklare och stegvis. Med rekommendation från Jan Björn på T-engineering togs beslutet att använda sig av en MOSFET-transistor i denna första drivkrets, vilket T-engineering använder sig av i sina styrsystem.

För att vända polariteten valdes en H-brygga. Då erfarenheter kring MOSFET-transistorer hade skapats i designen av den första drivkretsen togs beslutet att även konstruera h-bryggan utav MOSFET och inte en köpa en integrerad krets.

3.5 Utvecklingsmiljö för mjukvara till ECU-modulen

Utveckling av mjukvaran till ECU har skett modellbaserat i Simulink. Att använda detta verktyg var ett starkt önskemål från Chalmers Vera Team och motiverades bland annat med att det skulle förenkla vidareutveckling av systemet av studenter utan större kunskap om programmering. Simulink är en mjukvara, utvecklad av Mathwoks, som är tätt integrerad med MATLAB. MATLAB är en mer välkänd mjukvara, också utvecklad av Mathworks.

Med Simulink tillsammans med tilläggsprogramvaran Simulink coder, Embedded coder och ett för avsedd microkontroller anpassat blockset kan C-kod anpassad till mikrokontrollern genereras. Om även en avsedd kompilerare finns installerad kan koden direkt från Simulink kompileras och laddas upp till mikrokontrollern som sedan kör programmet. Sammantaget tillåter detta att simulering av, mätning på och felsökning av modellen först kan utföras i Simulink för att sedan laddas över och köras på mikrokontrollern.

Till Simulink finns två så kallade blockset anpassade till STM32F4-processorn, det ena utvecklat av STMicroelectronics och den andra, vid namn Waijung, av det thailändska företaget Aimagin. Ett blockset är en uppsättning funktionsblock som kan användas i Simulink och som motsvarar funktioner hos den mikrokontroller som de är avsedda för. Exempelvis finns i bägge ovan nämnda blockset block för inställning av digitala och analoga ingångar. I figur 4.13 i avsnitt 4.2.6 visas ett block för analoga ingångar.

Eftersom det var möjligt med Waijung att använda en gratis kompilerare samt hade fler funktioner som passade projektet så valdes Waijung. Till Waijung har också ett tilläggsblockset använts, Beyond Control, utvecklat av Antonio Flores på Robotics Lab vid universitet Carlos III i Madrid. Tillsammans gav dessa två blockset tillgång till många användbara funktioner hos mikrokontrollern.

3.6 Utveckling av styralgoritm till ECU:n

Här presenteras resonemang kring valda metoder för de mest fundamentala funktionerna i styralgoritmen. Styralgoritmen har utvecklats i Simulink, i avsnitt 4.2 beskrivs implementring av styralgoritmerna i simulinkmodellen i detalj.

3.6.1 Beräkning av tid till uppgift

För att kunna utföra uppgifter med en vinkelprecision högre än vad avståndet mellan vevaxelsensorns avkänningpunkter i sig tillåter behöver motorns position mellan dessa mätpunkter kunna uppskattas. Det här är viktigt för de positionskritiska uppgifterna tändning och insprutning.

En metod för att göra detta är att använda motorns momentana hastighet och sträckan kvar till uppgift i fråga och beräkna tiden kvar till att uppgiften skall utföras. Det behövs då en funktion för att bevaka tiden så att uppgiften kan utföras vid rätt tidpunkt.

I mikrokontrollern finns timermoduler som kan ställas att ge ett mjukvaruavbrott efter en utsatt tid, denna funktionalitet fanns dock inte i något av de två blockset som undersökts i projektet. För att komma runt detta problem har ett periodiskt avbrott används för att öka en räknare, och se om räknaren har uppnått tidigare fastställt räknarvärde då en uppgift skall utföras.

3.6.2 Hitta position

Till projektmotorn används två givare för att följa motorns position, en på kamaxeln och en på vevaxeln. Ingen missing tooth används, kamaxelsensorns signal utnyttjas istället som referenspunkt. Fördelen med det är att mindre beräkningar behöver göras varje gång vevaxelsensorn ger signal då det inte behöver avgöras om missing tooth har passerats eller inte. Kamaxelgivaren ger en referenspunkt som systemet sedan räknar positionen

från då vevaxelsensorn ger signal.

En nackdel med denna metod är att antalet referenspunkter också halveras till en per arbetscykel, därför ökas också maximala sträckan för synkronisering vid uppstart och återsynkronisering vid missad vevaxelsignal. Följderna av denna nackdel kan dock avhjälpas genom att placera den ensamma referenspunkten på lämpligt ställe. Eftersom styrsystemets positionskritiska uppgifter, insprutning och främst tändning, sker i varandra efterföljande takter, inom ett vevaxelvarv, så kan referenspunkten placeras strax framför den första av dessa uppgifter och samma funktionalitet som med två referenspunkter uppnås. Av dessa uppgifter är tändningen avsevärt mycket mer positionskänslig, därför har referenspunkten placerats strax innan tidigast möjliga tändposition.

Ett möjligt problem med denna metod uppkommer om kamaxelnsensorns avläsningspunkt placeras så att kamaxelsensorn ger signal samtidigt som vevaxelsensorn och detta ger en oförutsägbar ordningsföljd.

3.6.3 Bränslereglering

Eftersom projektmotorn saknar gasspjäll försvinner lastdimensionen ur bränslemappen. Bränslemappen relaterar i detta fall varvtal till volym luft in till förbränning för 20 jämt spridda varvtal mellan 0 och 7000 RPM. I mappen anges volymen luft som andel av full cylindervolym, fyllnadsgrad. Figur 4.29 på sidan 58 visar en bränslemapp från projektsystemet.

Ur bränslemappen uppskattas fyllnadsgraden, η_v, för aktuellt varvtal genom interpolation mellan närmste värden angivna i bränslemappen. Denna fyllnadsgrad konverteras till volym luft vid omgivningtryck genom multiplikation med cylindervolymen, V_d. Tillsammans med sensorvärden på omgivande lufttryck, P, och lufttemperatur, T, och specifika gaskonstanten, R, används detta värde till att via allmänna gaslagen beräkna massa luft som strömmat in till förbränningen(ekvation 3.1). Detta värde ger via inställt AFR_{stok} och lambdavärde, λ, massan bränsle som skall tillföras(ekvation 3.2).

$$m_{luft} = \eta_v * \frac{PV_d}{TR} \tag{3.1}$$

$$m_{bransle} = \frac{m_{luft}}{AFR_{stok} * \lambda} \tag{3.2}$$

3.7 Val av hårdvara till HMI

Valet av hårdvara baserade sig på den del av kravspecifikationen som inte direkt hade med styrning av motorn att göra, främst:

- Loggning av motordata under race.
- Display till föraren.

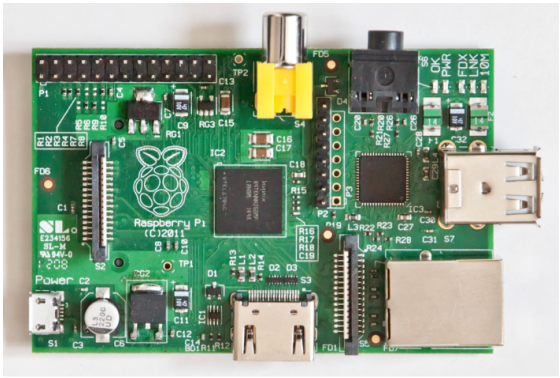
- Möjlighet att se livedata.

Valet att dela upp funktionaliteten i två sparata moduler innebar att kommunikationsmöjligheter mellan modulerna behövde tas hänsyn till. Basen för ECU-delen är en mikrokontroller vilket medförde att HMI-hårdvaran var tvungen att ha stöd för ett av kommunikationsprotokollen som stöds av den mikrokontrollern. En värsta falls-beräkning gjordes på den datamängd som antogs behöva skickas(ekvation 3.3).

$$\frac{\frac{7200\,varv\,per\,min}{60\,sek\,per\,min}}{2\,varv\,per\,cykel} = 60\,cykler\,per\,sekund = 60\,datapaket\,per\,sekund \tag{3.3}$$

Ekvation 3.3 visar att ingen stor mängd data kommer skickas, men det antogs ändå kräva en betydande mängd beräkningskraft från processorn. Beräkningen visar även att det troligtvis kommer genereras mycket loggdata, en förhållandevis stor lagringskapacitet kommer då krävas.

Enheten skulle också ha möjlighet att ansluta till en skärm, tillhandahålla information om geogafisk position och kopplas upp mot internet. Det sistnämnda en följd av kravet att man ska kunna se livedata. Hårdvaran fick inte heller vara för stor, väga mycket eller dra för mycket ström.



Figur 3.8: Raspberry Pi[3]

Baserat på resonemangen ovan valdes en enkortsdator av typen Raspberry Pi(Figur 3.8), RPi, som bas. RPi:n har en klockfrekvens på 700 MHz, 512 MB RAM och för lagring stödjer den SD-kort på upp till 32 GB. RPi har måtten 86 mm x 56 mm, väger 45 g, drar relativt lite ström och kostar runt 35 $. RPi är en välkänd enkortsdator med stor mängd användare runt om i världen.

Till RPi:n ansluts en höghastighets-UART-modul för att kommunicera med ECU:n och en USB Wifi-dongel för trådlös komunikation.

[3]Bildkälla: `http://commons.wikimedia.org/wiki/File:Front_of_Raspberry_Pi.jpg`

3.8 Utvecklingsmiljö för HMI

Efter en snabb undersökning av operativsystem till RPi:n valdes Raspbian. Raspbian är det officiella operativsystemet och har störst utvecklarcommunity, vilket medför minskade utvecklings- och felsökningstider. Raspbian innehåller mer funktionalitet än vad som krävdes för projektet men gav möjligheten att snabbt komma igång för att sedan analysera vilken funktionalitet som är överflödig. En fördel med att ha ett välkänt operativsystem i grunden är också möjligheten att använda redan färdigutvecklad programvara.

VNC- och SSH-servrar installerades och RPi:n kopplades upp till internet. En VNC-server möjliggör fjärråtkomst till skärmen vilket innebär att utvecklingen kan ske på distans samtidigt som man kan se användargränssnittet som visas på skärmen på RPi:n. En SSH-server ger fjärråtkomst till filsystemet och terminalen vilket ger tillgång till källkodsfilerna och möjliggör kompilering på distans. Möjligheten att arbeta på distans var ett krav för att flera skulle kunna utveckla produkten samtidigt och detta mot en fysisk enhet som inte går att dela.

Som versionshantering och backup av källkoden valdes Git. Valet grundades på att Git är ett av de största versionshanteringsystemen och även det som lärs ut på Chalmers. Som värd för projektet valdes github.com eftersom det är en välkänd plats för lagring av Git-projekt och tillförlitligheten är hög. Studenter har även möjligheten att gratis få tillgång till 5 privata projekt.

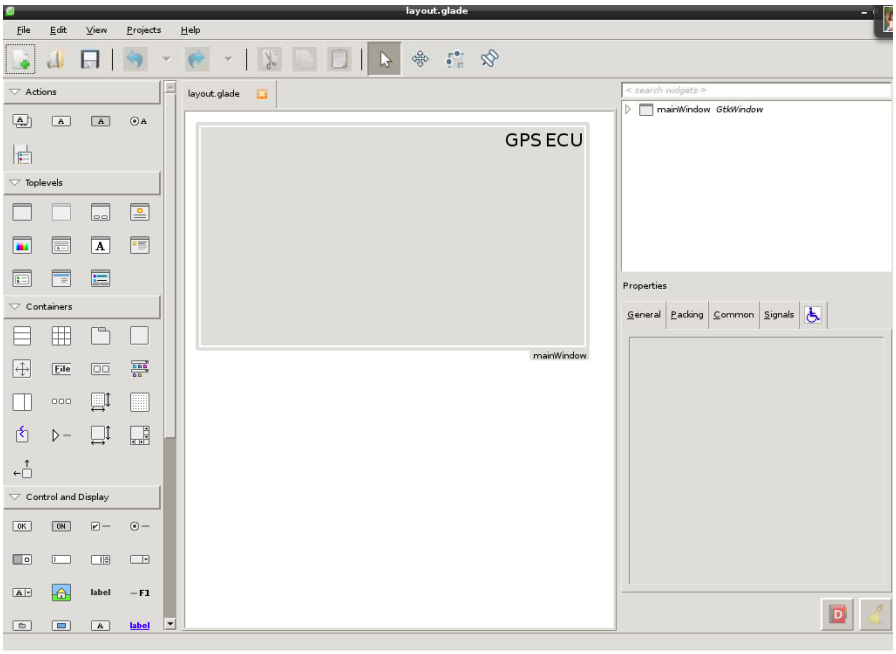
För att på ett effektivt sätt kunna hantera stora mängder loggdata valdes en lokal MySQL-databas för lagring. Långa svarstider är inget problem i detta fall eftersom databasen körs lokalt. För att lagra datan krävs det mer beräkningskraft per loggdata-enhet jämfört med att lagra direkt i textfiler. Detta problem kan dock minskas genom att samla ihop ett antal loggdataposter och sedan lägga in alla samtidigt, så kallad batching.

3.9 Val av programmeringsspråk för HMI

För att få ut optimal prestanda togs beslutet att använda ett maskinnära programmeringsspråk. Ett färdigt C++ bibliotek till 3G/GPS modulen fanns redan, detta samt fördelarna med ett språk som stödjer objektorienterad programmering motiverade att C++ valdes.

Användargränssnittet byggdes med hjälp av biblioteket Gtkmm. Detta bibliotek valdes för att det är det officiella C++-gränssnittet till C-biblioteket GTK+. GTK+ i sin tur är ett välkänt bibliotek för att skapa användargränssnitt och är bland annat en av huvudkomponenterna i GNOME men används även i program som Firefox. C++ biblioteket är klassbaserat, kommer med färdiga komponenter och är även crossplatform vilket ökar flexibiliteten för framtida utveckling.

Att konstruera användargränssnitt i källkod genererar stora, hårdkodade och svårlästa textblock och därför valdes istället att använda Glade(Figur 3.9). Glade är ett verktyg som ger möjligheten att grafiskt designa användargränssnittet till GTK+. Resultatet sparas ner i en XML-formaterad fil som läses in av C++ programmet vilket ökar flexibiliteten avsevärt och sparar utvecklingstid.



Figur 3.9: Glade användargränssnitt

3.10 Livedata

För att ha möjlighet att uppdatera data live införskaffades en 3G/GPS-modul från Cooking Hacks[10], denna ansluts direkt på RPi:n. Som namnet antyder kan den både koppla upp mot 3G-nätet samt använda GPS för att räkna ut geografiska kordinater och hastighet. Att ansluta 3G/GPS-enheten direkt på RPi:n via GPIO ger bara en enhet och väldigt smidig implementation via textbaserade kommandon, AT-kommandon.

Presentationen av livedatan valdes att läggas på Chalmers Vera Teams hemsida, chalmersverateam.se. Webbhotellet som används ligger på one.com och innehåller även en MySQL-databas där livedatan kan lagras. Sidan är uppbyggd i Wordpress och tanken var därför från början att både presentation och uppdatering av data skulle ske med detta. Wordpress är ett simpelt verktyg att använda vid presentation av data, däremot om datan ska uppdateras kontinuerligt med ny data visade sig Wordpress vara ett krångligt verktyg. Därför valdes istället att i grunden ha Wordpress men låta all visualisering och uppdatering skötas av JavaScript. För att visualisera data används två färdiga bibliotek,

Google Maps JavaScript API v3 för att rita upp kartor och NVD3 för att skapa grafer. Dessa bibliotek valdes därför att de är användarvänliga och har bra dokumentation med många exempel.

3.11 Programmering av mappar

För att kunna producera ett enhetligt och användarvänligt styrsystem beslutades att ECU-modulens inställningsbara parametrar skulle programmeras via HMI-delen. För att minimera kravet på extern hårdvara och mjukvara valdes att programmeringen skulle ske via en lokal webbserver på HMI-modulen. För att programmera och avläsa värden på ECU-modulen behövs då endast en webbläsare på den enhet man programmerar med. Även loggdatan kommer att presenteras via detta gränssnitt.

För att ansluta till den lokala webbservern har ett trådlöst nätverkskort anslutits via USB till RPi:n. Den sattes upp som en lokal accesspunkt vilket leder till att programmeringen av ECU-modulen och visning av loggdata kan ske utan någon fysisk anslutning till Vera. Vera kommer alltså inte behöva tas isär för att göra inställningar eller se loggdata. Att kravet enbart är att enheten har Wi-Fi och en webbläsare innebär också att endast en surfplatta eller någon annan smidigare enhet behöver tas med till race.

Värdena på mapparna som ställs in i gränssnittet kommer skickas till ECU-modulen där det lagras i ett icke-flyktig flashminne. Detta gör att ECU-modulen kan köras helt frånkopplad från HMI-modulen om det skulle behövas.

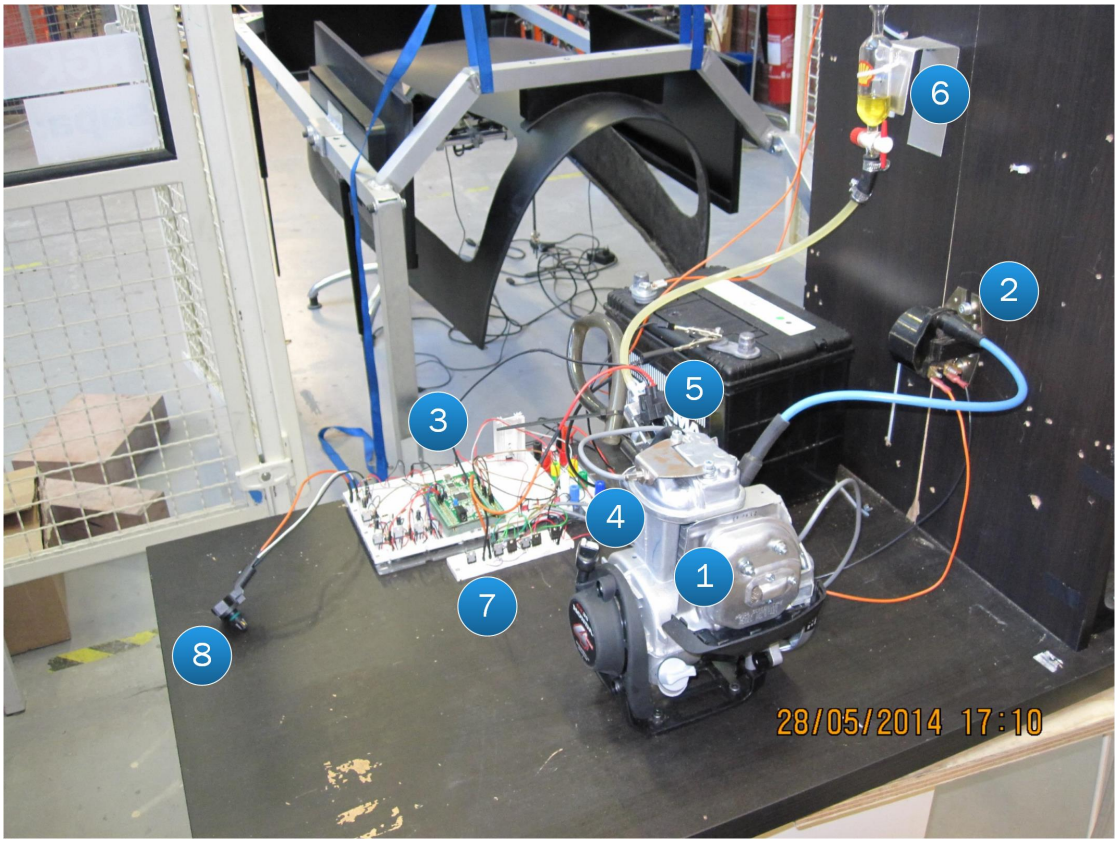
3.12 Gränssnitt till föraren

I syfte att kartlägga vilken typ av information som är relevant för Veras förare intervjuades denne i ett tidigt skede. Det visade sig finnas stor potential att ge föraren nya verktyg för att optimera sin körning, tid och hastighet var den enda information som föraren då hade att tillgå. Föraren uttryckte önskemål om att visa antal körda varv och att funktioner som medelhastighet och varvtid även skulle uppskattas. Detta ansågs vara bra att börja med för att sedan kunna vidareutveckla. Föraren beskrev också att det är många olika moment och mycket att tänka på under ett race, därför var ett enkelt och tydligt användargränssnitt önskvärt.

För att kunna hålla koll på antal varv och varvtider under race diskuterades olika lösningar. I kandidatarbetet VerApp[1] implementeras denna funktion med hjälp av GPS. Andra lösningar som att ha sändare vid målgången och en mottagare i bilen diskuterades också. Att implementera funktionen med knappar valdes, fler reflektioner kring detta val diskuteras under kapitlet Diskussion. Att använda knappar betyder att varje gång Vera passerar startlinjen tycker föraren på en knapp. Då räknas antal varv och varvtiden ut och visas för föraren på displayen.

3.13 Testrigg

För att verifiera styrsystmet byggdes en testrigg med exprimentmotorn upp, Figur 3.10.



Figur 3.10: Testrigg för verifiering av styrsystemets funktion. Honda GX35(1), tändspole(2), STM32F4 Discovery(3), kamaxelsensor(4), spridare(5), bränsletank(6), drivelektronik för tändspole och spridare(7), MAP-sensor(8).

Första testerna gjordes med förgasare istället för spridare, tändningen kunde då testas separat. När tändningsregleringen verifierats kopplades spridare med trycksatt bränsle in och tester av bränsleregleringen kunde utföras.

4. Resultat

I detta kapitel presenteras projektets resultat.

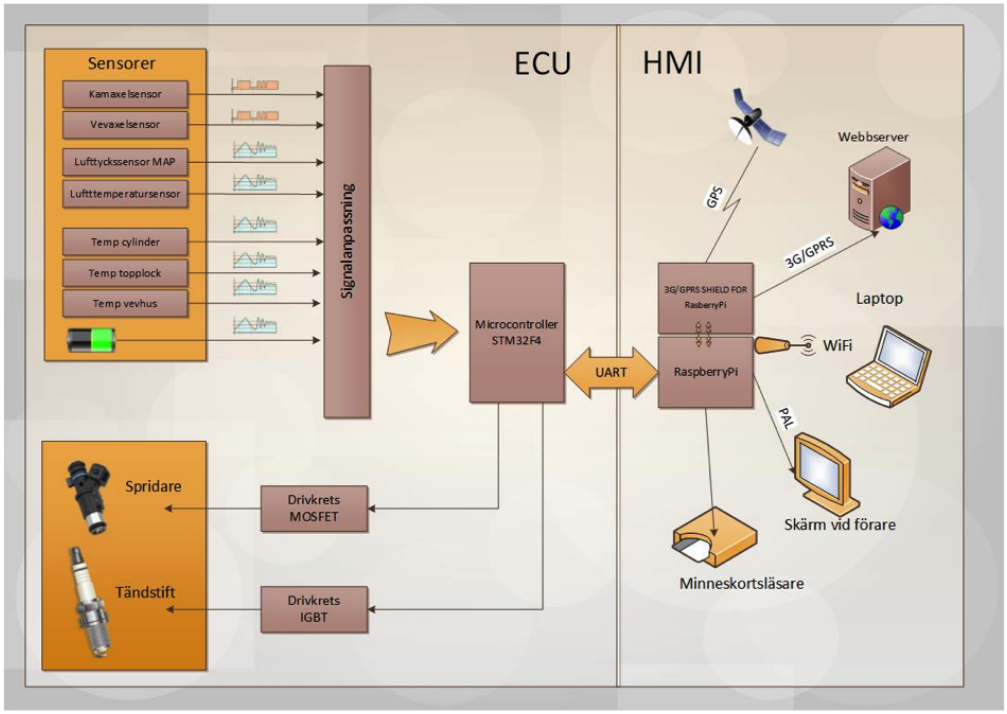
4.1 Överblick av styrsystemet

Styrsystemet illustreras i Figur 4.1 och är uppbyggt av två moduler, ECU-modulen och HMI-modulen.

- ECU-modulen bygger på en mikrokontroller, STM32F4VG07.
- HMI-modulen bygger på en enkortsdator, Raspberry PI med operativsystemet Raspbian.

I systemet ingår följande sensorer:

- Kamaxelsensor: Hall Effect Proximity Switch CYKN8-02CL0.
- Vevaxelsensor: Hall Effect Proximity Switch CYKN8-02CL0.
- Temperatursensorer: LM35.
- Luftryckssensor(MAP): BOSCH 0 261 230 031.

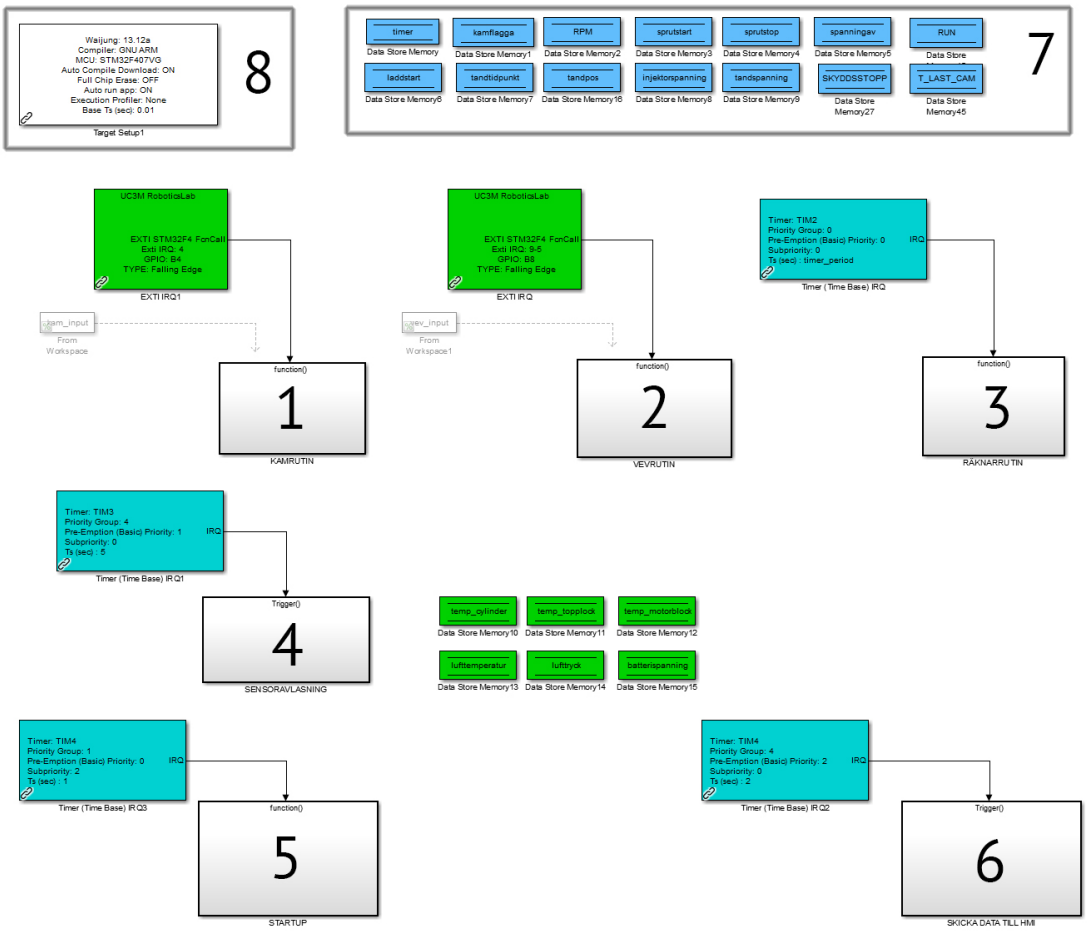


Figur 4.1: Illustration av styrsystemet

ECU-modulen tar emot signaler från sensorerna och använder dessa för att styra tändning och insprutning i motorn. ECU-modulen skickar även data till HMI-modulen som bland annat sköter loggning av data, presentation av data till föraren, presentation av data på internet via 3G och hanterar GPS. Modulerna kommunicerar med varandra via UART.

4.2 Simulinkmodell

Modellen delas in i delmodeller, subsystem, som antingen exekveras periodiskt, med olika periodtider, eller då signal inkommer utifrån. Nedan presenteras en överblick av modellens mest centrala delar och deras funktion.



Figur 4.2: Simulinkmodell, toppnivå

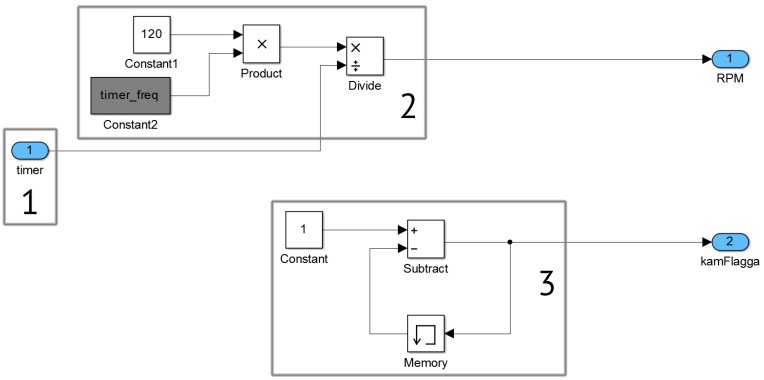
Lista över subsystem från toppnivå i modellen. Numren i listan refererar till numren i Figur 4.2.

1. Kamrutin. Exekveras då kamsensorn ger signal(interuptstyrd) Beskrivs i avsnitt 4.2.1.
2. Vevrutin. Exekveras då vevsensorn ger signal(interuptstyrd) Innehåller i sin tur ett större antal subsystem, största subsystemet i modellen. Beskrivs i avsnitt 4.2.2.
3. Räknarrutin. Exekveras på tidsbas, kortaste tidsbasen i modellen. Beskrivs i avsnitt 4.2.3.
4. Sensoravläsning. Exekveras på tidsbas och läser in värden från styrsystemets analoga sensorer. Beskrivs i avsnitt 4.2.4.
5. Periodisk hjälprutin. Exekveras på tidsbas, första gången denna körs laddas parametrar in från flashminnet i en startrutin. Beskrivs i avsnitt 4.2.5.
6. Loggrutin. Exekveras på tidsbas, skickar loggdata till HMI-modulen. Beskrivs i avsnitt 4.2.6.

Minnesblocken(7) används för att spara och skicka data mellan olika subsystem. Target setup(8) är ett block i Waijung och används för att konfigurera och anpassa kodgenereringen till målprocessorn, STM32F4VG07.

4.2.1 Kamrutin

Kamrutinen visas i figur 4.3 och exekveras som ett avbrott då signal ges från kamaxelsensorn. Kamrutinen tar in räknarens värde, figur 4.3(1)(räknaren beskrivs i avsnitt 4.2.3). Då räknaren nollställs efter kamrutinen körts representerar räknarens värde antalet räknarperioder som gått sedan förra gången kamrutinen kördes. Räknarvärdet används för att beräkna medelhastigheten över föregående arbetscykel, 4.3(2). Detta värde sparas i ett minnesblock(ej i bild), i enheten RPM då denna används i mapparna.



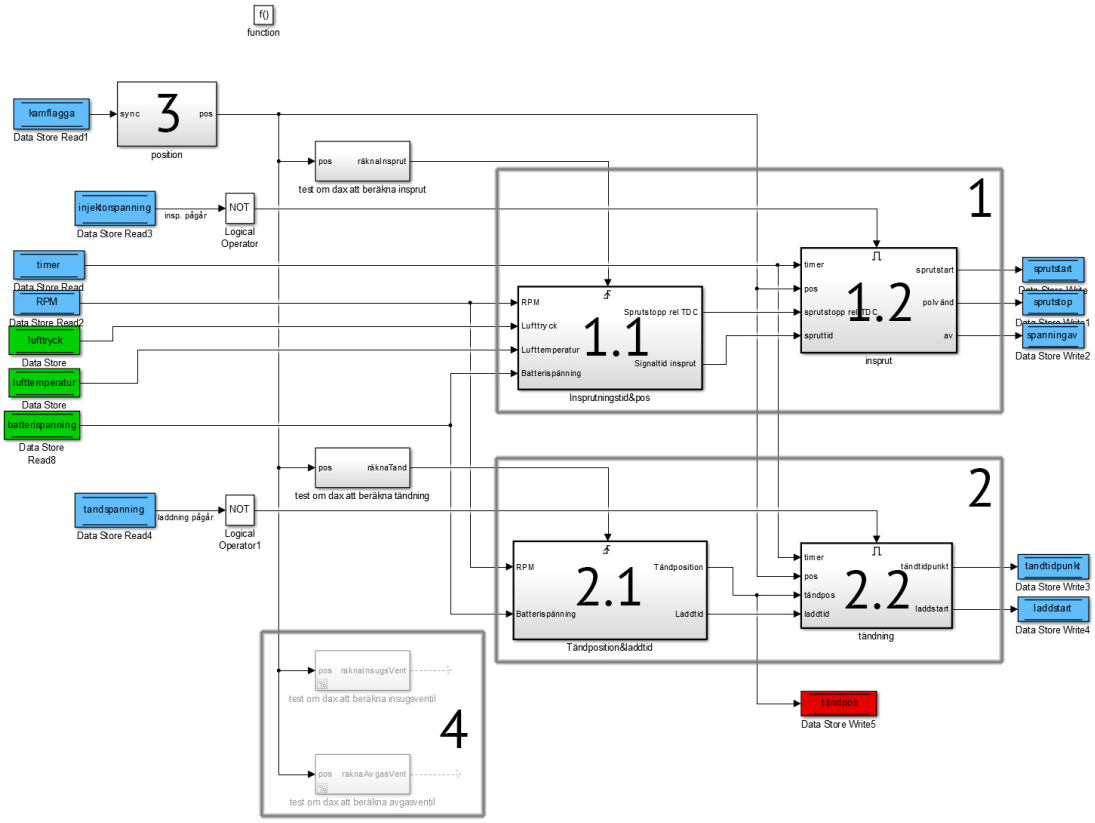
Figur 4.3: Kamrutin

Kamrutinen skiftar också värdet på en variabel, sparad i ett annat minnesblock(ej i bild) mellan 0 och 1, 4.3(3). Detta värde benämns kamflaggan och används i andra subsystem

för att se att kamsignal har kommit sedan förra gången de exekverades. Exempelvis får räknarrutinen information om när den skall nollställa räknarvariabeln via detta minnesblock.

4.2.2 Vevrutin

Vevrutinen visas i Figur 4.4) och utförs som ett avbrott då signal ges från vevaxelsensorn, 48 gånger per arbetscykel. I denna rutin sker uppdatering av motorns position och beräkningar av tidpunkter i arbetscykeln för vilka signaler till tänd- och bränsleinsprutningssystemen skall skickas.



Figur 4.4: Vevrutin

1. Beräkning av tidpunkter för bränslesystemet
    1.1. Bränsleberäkning, beräkning av position för sprutstopp och öppentid för spridare
    1.2. Konvertering till tidpunkter för spridare
2. Beräkning av tidpunkter för tändsystemet

2.1. Tändberäkning, beräkning av position för gnista och laddtid för spolen

2.2. Konvertering till tidpunkter för tändsystemet

3. Positionsuppdatering

4. Förberedelse för framtida tidskritiska arbetscykel-uppgifter

Beräkningarna av tidpunkterna för tändning, Figur 4.4(1), respektive insprutning(2) sker i två steg, (1.1 & 1.2) respektive (2.1 & 2.2). I första steget beräknas signaltid och motorns position då signal skall ges. Dessa beräkningar sker en gång per arbetscykel. I andra steget används de värden som gavs från steg ett, dessa konverteras till tidpunkter under arbetscykeln, uttryckta i värden hos räknarrutinens(avsnitt 4.2.3) uppräkningsvariabel. Dessa beräkningar sker varje gång vevrutinen exekveras. Det första steget i beräkningen behöver bara utföras en gång per arbetscykel då det värde in till beräkning som uppdateras mest frekvent är varvtalet och detta uppdateras en gång per arbetscykel. Det andra steget utförs lika ofta som värdet på momentanhastigheten uppdateras, vid varje ny vevaxelsensorsignal.

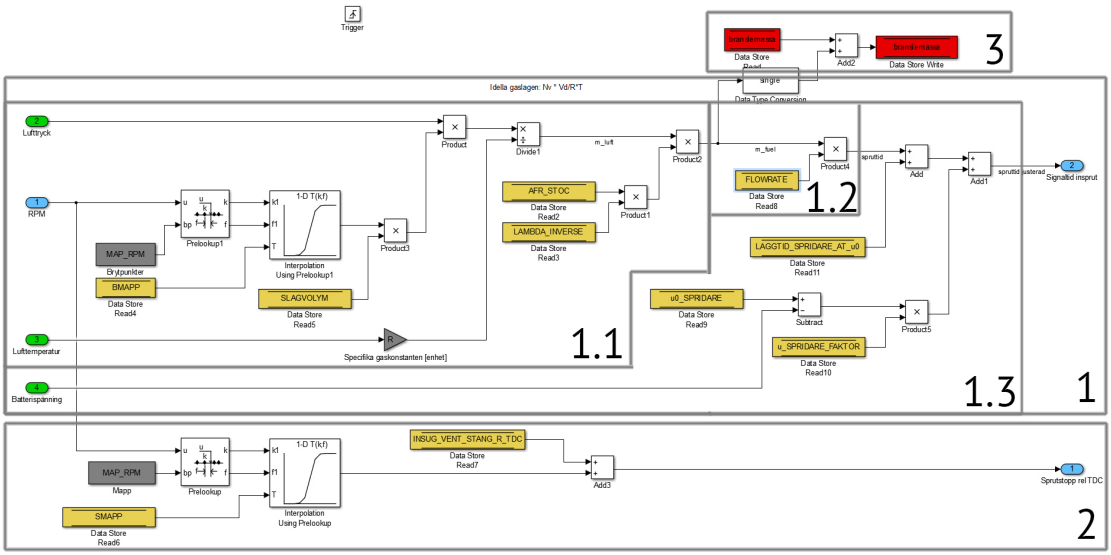
Bränsleberäkning Två beräkningar utförs i detta subsystem. I den första(Figur 4.5(1)) beräknas den tid systemet skall hålla spridaren öppen. I ett första steg i denna beräkning(1.1) används varvtal, lufttryck, lufttemperatur och bränslemappen för att beräkna massan bensin som skall tillföras nästkommande förbränning, denna beräkning beskrivs i avsnitt 2.4. I nästa steg(1.2) används en spridarkonstant(eng. *flowrate*), ϕ, för att omvandla bränslemassa, m_{brans}, till spridartid, t_{okomp} enligt ekvation 4.1.

$$t_{okomp} = \phi * m_{brans} \tag{4.1}$$

Ekvation 4.1 gäller när spridaren är fullt öppen, då spridaren inte ger sama flöde vid öppning och stängning görs också en kompensering för detta(1.3). Öppnings- och stängningstiden har antagits påverkas av batterispänningen och kompensation utförs enligt ekvation 4.2.

$$t_{komp} = t_{okomp} + K_{u0} + (u - u_0) * P_{u0} \tag{4.2}$$

Ekvation 4.2 ger kompenserad spridartid, t_{komp}. K_{u0} representerar kompensering vid nominell spänning, u_0 nominell spänning, u uppmätt spänning och P_{u0} en proportionallitetskonstant.

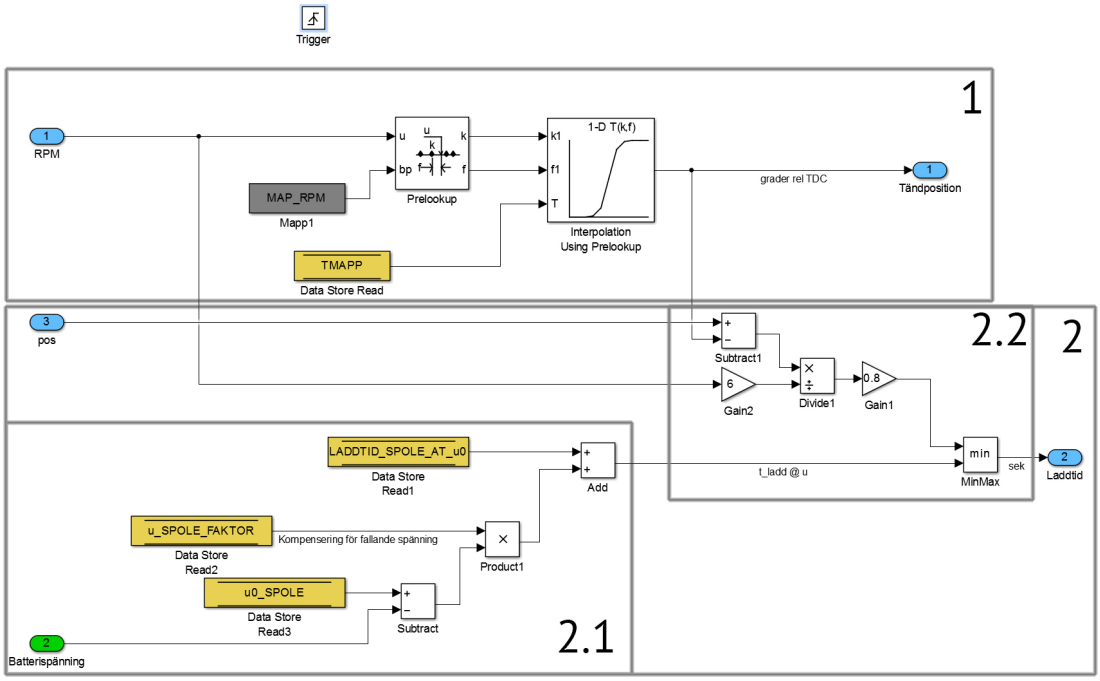


Figur 4.5: Rutin för bränsleberäkning

I den andra beräkningen(4.5(2)) interpoleras den vinkel för vilken spridaren skall stängas fram ur en mapp. Denna mapp innehåller värden på önskad stängvinkel för 20 jämnt spridda varvtal emellan 0 och 7000. Värden i mappen är angivna relativt insugsventilens stängningsvinkel.

Beräknad bränslemassa adderas också till värdet i ett minnesblock som håller ackumulerad bränslemassa(4.5(3)) detta värde loggas av loggrutinen som beskrivs i avsnitt 4.2.6.

Tändberäkning I detta subsystem, figur 4.6, utförs två beräkningar, den första ger tändposition och den andra laddtiden för tändspolen.



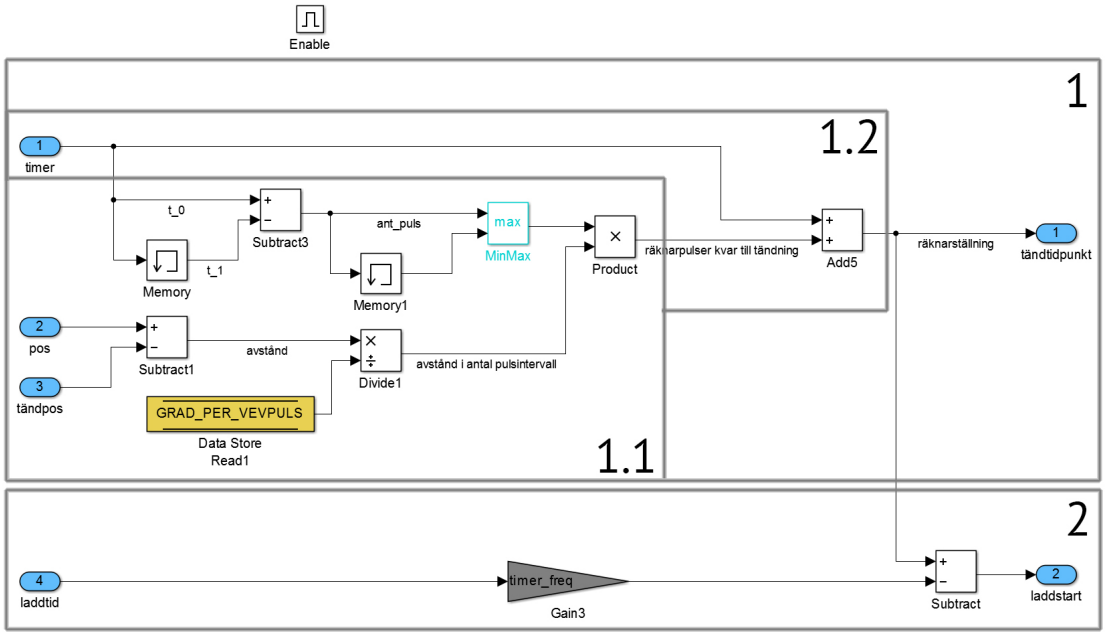
Figur 4.6: Tändberäkningsrutin

Beräkningen för tändposition(4.6(1)) tar in varvtalet vilket används för att interpolera fram önskad tändposition ur tändmappen.

Beräkningen för laddtid(4.6(2)) tar in batterispänning. Förhållandet mellan batterispänningen och laddtiden för att uppnå önskad laddning i spolen har linjäriserats kring nominell batterispänning. Första steget i beräkning plockar fram önskad laddtid vid aktuell batterispänning(2.1). I nästa steg säkerställs att ett för stort värde inte sätts(2.2), om beräknat värde i steg ett är för stort för att laddning skall hinnas med vid aktuellt varvtal så väljs istället största möjliga värde för detta varvtal.

Konvertering till tidpunkter för bränslesystemet Konverteringen till tidpunkter för bränslesystemet utförs analogt med beräkningen för tändsystemet, där intervallet laddtid motsvaras av spruttid och positionen för tändning motsvaras av positionen för sprutstopp. Konvertering till tidpunkter för tändsystemet beskrivs nedan.

Konvertering till tidpunkter för tändsystemet I detta subsystem, figur 4.7, beräknas tidpunkter för tändning(4.7(1)) och laddstart(4.7(2)), tidpunkter då ECU-modulen skall slå på respektive av spänning över tändspolen. Tidpunkterna som beräknas av detta subsystem ges i räknarställning och används i räknarrutinen(avsnitt 4.2.3).



Figur 4.7: Konvertering till tidpunkter för tändsystemet

Denna rutin och rutinen för positionsuppdatering(avsnitt 4.2.2) är båda subsystem till vevrutinen(Figur 4.4) och utförs därför båda periodiskt var 15:e vevaxelgrad. Rutinen för positionsuppdatering utförs först, detta innebär att variabeln som håller motorns position är uppdaterad varje gång denna rutin körs. Vevaxelns position används i detta subsystem tillsammans med önskad tändposition under aktuell arbetscykel(beräknad i tändberäkningsrutinen(avsnitt4.2.2)) för att beräkna antalet vevaxelgrader kvar till tändning. Aktuellt räknarvärde(avsnitt 4.2.3) tas också in och sparas. De två senaste räknarvärdena används för att beräkna antalet räknarpulser sedan förra gången detta subsystem kördes, 15 vevaxelgrader tidigare. Önskad laddtid under aktuell arbetscykel i sekunder(också beräknad i tändberäkningsrutinen) tas också in och används för att beräkna tidpunkt för laddstart(4.7(2)).

För att beräkna tändtidpunkten används motorns nuvarande position, önskad tändposition och räknarvärdet. Beräkningen sker i två steg. Steg ett(4.7(1.1)) ger uppskattad räknaröknin till tändning(ekvation 4.3). Avståndet kvar till tändning i grader, Δs beräknas. Denna divideras med antalet grader per vevsensorsignal, Θ_n, och avståndet kvar

till tändning givet i antal vevsensorintervall fås. Detta multipliceras med antalet räknarpulser under förra vevgivarintervallet, Δt, och uppskattat räknarökning till tändning fås, t_{tand_rel}.

$$t_{tand_rel} = \frac{\Delta s}{\acute{v}} = \{\acute{v} = \frac{\Theta_n}{\Delta t}\} = \frac{\Delta s \Delta t}{\Theta_n} \tag{4.3}$$

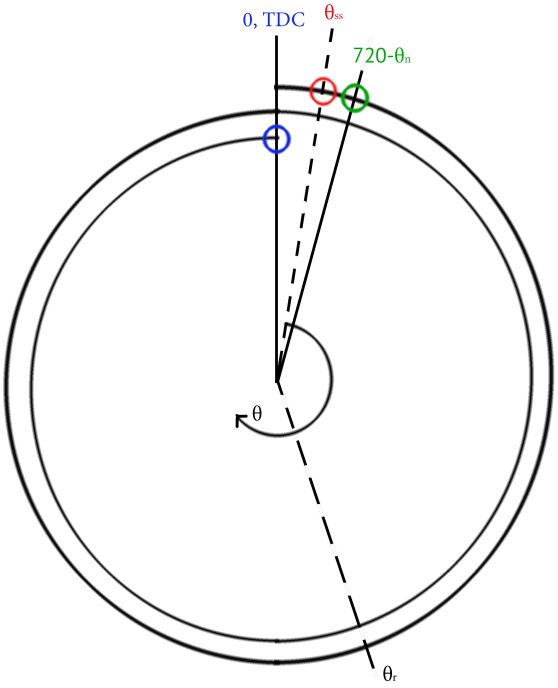
I beräkningen antas att motorn kommer fortsätta i samma hastighet som den hade mellan de senaste avkänningspunkterna på vevaxeln, passerad tid mellan denna och föregående sensorsignal från vevaxelsensorn används, Δt. Då detta utförs vid varje vevsensorsignal så säkerställs att när uppgiften väl utförs så har senaste värde på medelhastigheten över ett 15-gradersintervall, $\acute{v}$, används i beräkningen. Detta är önskvärt då motorns hastighet förändras under arbetscykeln.
I setg två(4.7(1.2)) adderas uppskattad räknarökning, t_{tand_rel} från ekvation 4.3, till nuvarande räknarställning, t_{abs}, och slutresultatet blir den räknarställning vid vilken systemet skall ge tändsignal, t_{tand_abs}(ekvation 4.4).

$$t_{tand_abs} = t_{abs} + t_{tand_rel} \tag{4.4}$$

Tidpunkten för laddstart beräknas genom att laddtiden i sekunder, t_{ladd} i ekvation 4.5, multipliceras med räknarfrekvensen, ω, detta ger laddtiden i räknarökning, t_{ladd_rel}. Eftersom laddningen av spolen avslutas då tändningen sker subtraheras räknarökningen under laddning från tändtidpunkt, t_{tand_abs}, och tidpunkten för laddstart fås, t_{ladd_abs}(ekvation 4.5).

$$t_{ladd_abs} = t_{tand_abs} - t_{ladd_rel} = t_{tand_abs} - t_{ladd} * \omega \tag{4.5}$$

Postionsuppdatering Denna rutin är ett subsystem till vevrutinen(avsnitt 4.2.2) och körs därför när vevaxelsensorn ger signal. Systemet uppdaterar motorns position, Θ(Figur4.8), i arbetscykeln. Motorns arbetscykel är 720 vevaxelgrader, systemets positionsvariabel kommer därför alltid befinna sig i ett intervall mellan 720 och 0, $\Theta \in (720, 0]$. TDC mellan kompressionstakten och arbetstakten representeras i systemet av $\Theta = 0$.



Figur 4.8: Vevaxelposition i arbetscykeln

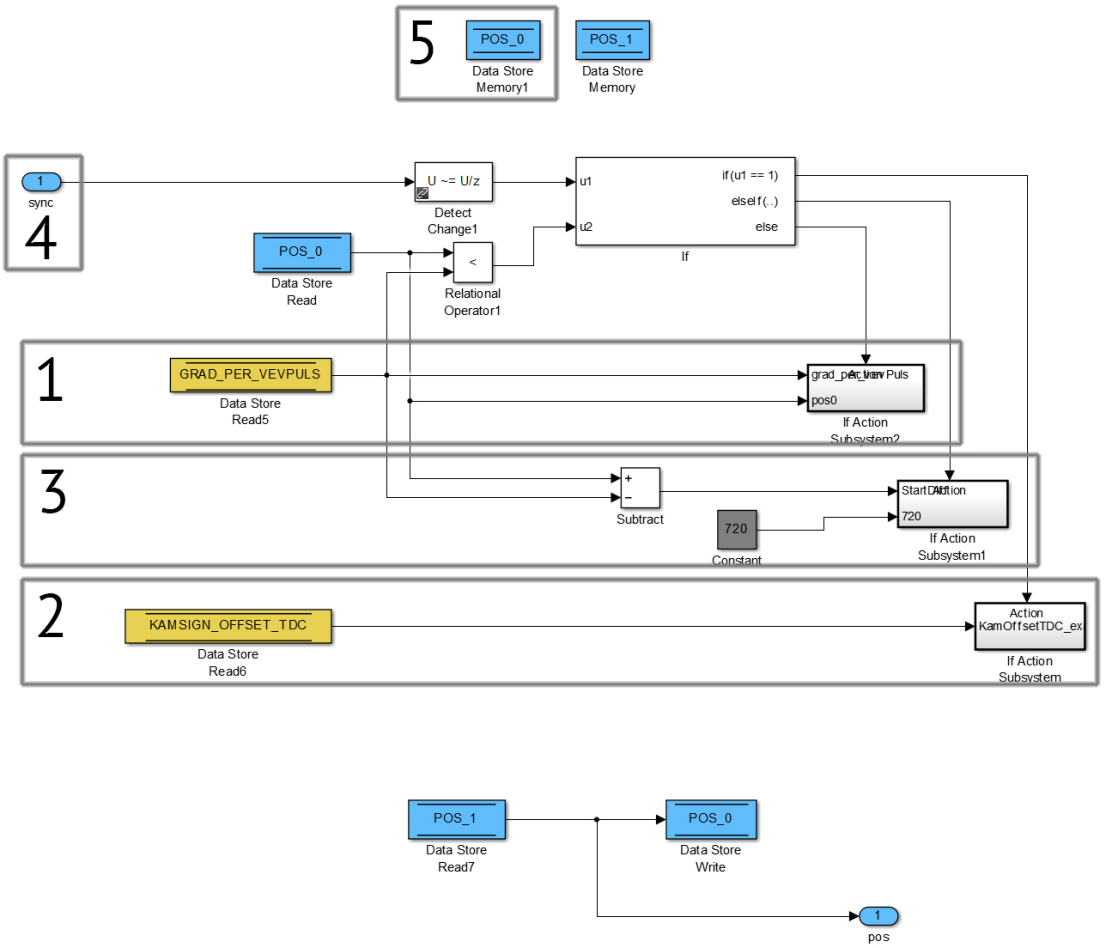
Positionsuppdateringen beskrivs av ekvation 4.7. Systemet räknar ner motorns position från ett startvärde, Θ_{ss}(Figur 4.8), i steg om, Θ_n, tills att föregående positionsvärde, Θ_i, är mindre än Θ_n. Då sätts det nya värdet, Θ_{i+1}, åter till starvärdet(4.7(3)). Θ_n representerar vevaxelvinkeln mellan två avkänningspunkter för vevaxelsensorn(ekvation 4.6). Vid en punkt under arbetscykelns 720 vevaxelgrader ger kamaxelsensorn signal. Detta subsystem tar in kamrutinens(avsnitt 4.2.1) kamflagga(Figur 4.9(4)), denna indikerar om kamsensor har gett signal sedan förra gången subsystemet kördes. Om detta har inträffat synkroniseras positionen. Positionsvariabelns värde sätts då till den bestämda vinkel, Θ_r, där vevaxeln skall befinna sig när kamsignal har inkommit. I ekvation 4.7 representeras att kamflaggan har ändrats av $kam = 1$.

$$\Theta_n = \frac{360}{N} \tag{4.6}$$

N i ekvation 4.6 representerar antalet avkänningspunkter för vevaxelsensorn. Avkänningspunkterna är jämnt spridda. På projektmotorn är $N = 24$.

$$\Theta_{i+1} = \begin{cases} \Theta_i - \Theta_n & \text{om kam} \neq 1 \text{ och } \Theta_i \geq \Theta_n \quad (1) \\ \Theta_r & \text{om kam} = 1 \quad (2) \\ 720 - (\Theta_n + \Theta_i) & \text{om kam} \neq 1 \text{ och } \Theta_i < \Theta_n \quad (3) \end{cases} \tag{4.7}$$

Θ_i((Figur 4.9(5)) sparas internt i subsystemet och Θ_r är en inställning i styrsystemet(värdet mäts upp).



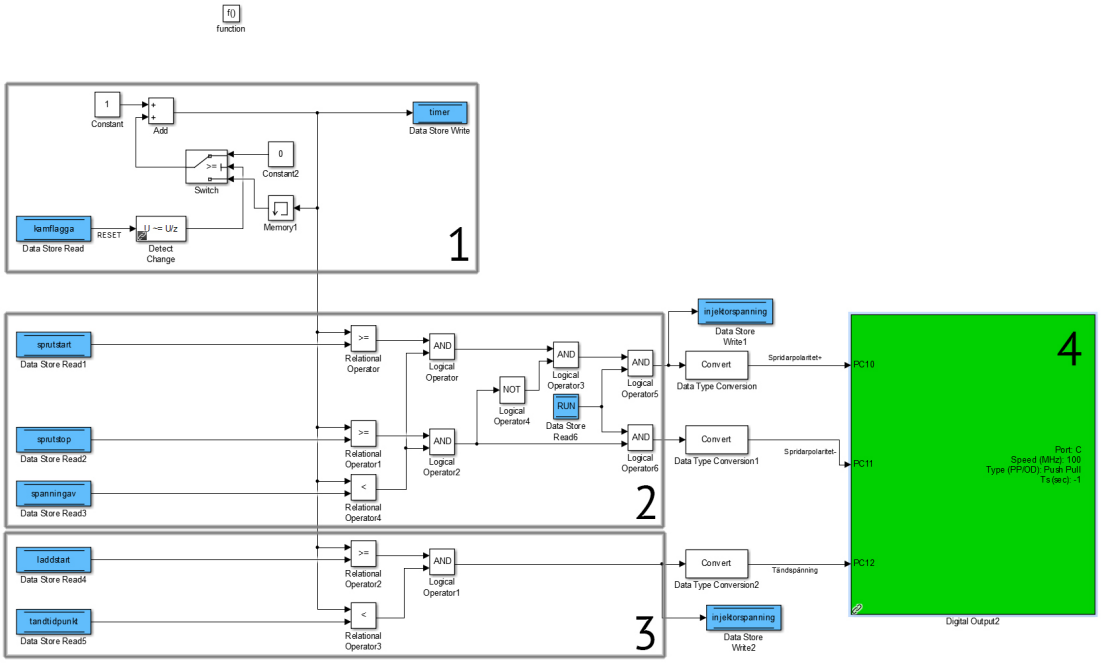
Figur 4.9: Rutin för positionsuppdatering

Markering 1 till 3 i figur 4.9 motsvarar utfall 1 till 3 i ekvation 4.7.

Förberedelse för framtida tidskritiska arbetscykel-uppgifter I figur 4.4 på sidan 32 visas två subsystem(4.4(4)) i vevrutinen. Dessa subsystem är i dagsläget bortkommenterade och har ingen funktion. I framtiden om exempelvis solenoidstyrda ventiler utvecklas till motorn kan beräkningar för deras funktion utföras efter samma mall som beräkningarna för bränslesystemet och tändsystemet, subsystemen är en förberedelse för detta.

4.2.3 Räknarrutin

Detta subsystem exekveras periodiskt med hög frekvens, denna frekvens avgör med vilken vinkelprecision tändning och bränsleinsprutning kan ske.



Figur 4.10: Räknerutin

Systemet har tre funktioner, dels att räkna och uppdatera förlöpt tid under arbetscykleln(Figur 4.10(1)), dels att testa om tidpunkten för tändning(4.10(2)) och insprutning(4.10(3)) är inne och dels att om så är fallet skicka ut signal. Tid i arbetscykeln hålls i ett minnesblock i enheten räknarperiod, den period med vilken räknarrutinen exekveras, tiden ökas med ett varje gång systemet exekveras. Om Kamflaggans(se avsnitt 4.2.1) har ändrats sedan förra gången systemet kördes så har kamsensorsignal inkommit och då nollställs istället räknaren.

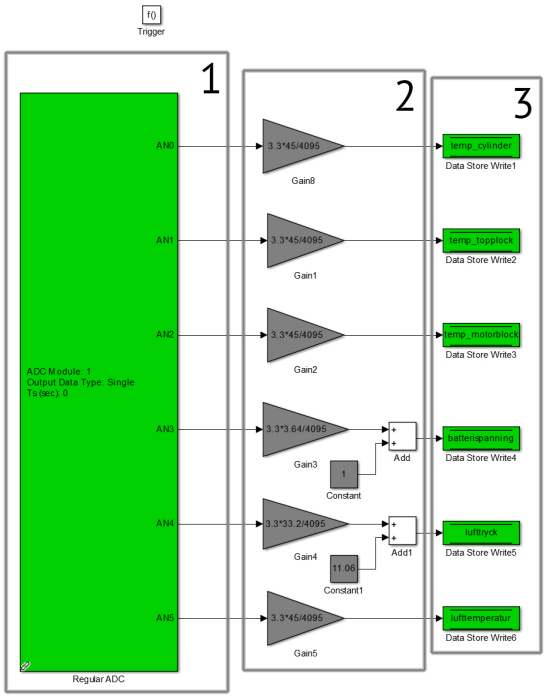
Varje gång systemet exekveras, efter att räknaren har inkrementerats, testas om räknaren har uppnått värde för någon tidskritiskt händelse i systemet, dessa listas nedan.

- Öppna spridare - Spänning över spridare med positiv polaritet.
- Stäng spridare - Spänning över spridare med negativ polaritet.
- Spridare stängd - Ingen spänning över spridaren.
- Starta laddning av tändspole - Spänning över tändspole på.
- Ge gnista - Spänning över tändspole av.

Testerna för spridaren(4.10(2)) får beräknade tidpunkter från rutinen som konverterar till tidpunkter för bränslesystemet, denna rutin beskrivs i avsnitt 4.2.2. Testerna för tändspolen(4.10(3)) får beräknade tidpunkter från motsvarande rutin för tändsystemet(4.2.2). Om räknarvärde där en händelse skall utföras har uppnåtts skickas signal för detta på en pin på mikrokonrollern. Det gröna blocket i bilden(4.10(4)) är ett digitalt IO-block och används till att sätta pinnar på mikrokontrollern, valda pinnar kan utläsas i figuren.

4.2.4 Sensoravläsning

Denna rutin exekveras var 5:e sekund och uppdaterar sensordata i systemet. Blocket till vänster(1) i figur 4.11 representerar AD-omvandlaren. Inlästa sensorvärden från AD-omvandlaren skalas om till rätt enhet(4.11(2)) och sparas i minnesblock(4.11(3)) som kan kommas åt från övriga delar av modellen.

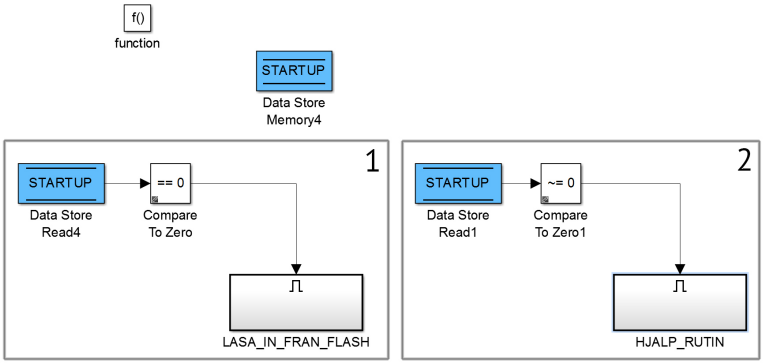


Figur 4.11: Sensoravläsning

På AD-omvandlarblocket(4.11(1)) kan spännignen över valda ingångar på mikrokontrollern utläsas. För att få spänningen, utryckt i Volt, som ligger på ingången multipliceras värdet med 3,3/4095. Detta på grund av att AD-omvandlaren har 12-bitars upplösning($2^{12} = 4096$) och tar in värden mellan 0 V till 3,3 V. Övriga faktorer som syns i 4.11(2) skalar signalen för att kompensera för de signalanpassningskretsar som redovisas i avsnittet 4.3.

4.2.5 Periodisk hjälprutin

Detta subsystem exekveras periodiskt med en periodtid på 2 sekunder. Första gången denna rutin exekveras efter uppstart körs en uppstartsrutin(Figur 4.12(1)). När detta en gång är utfört kan andra funktioner som behöver exekveras periodiskt köras, de placeras då i det andra subsystemet(4.12(2)). Tanken är att i detta subsystem utföra tidsperiodiska uppgifter som inte har med kommunikation eller sensoravläsning att göra.

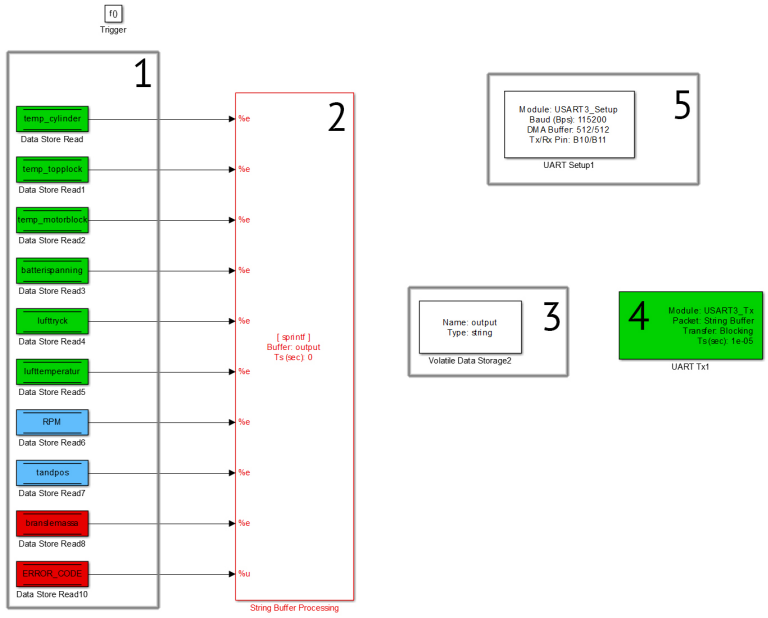


Figur 4.12: Periodisk hjälprutin

Uppstartsrutin
Detta subsystem(Figur 4.12(1)) ligger under den periodiska hjälprutinen och exekveras första gången hjälprutinen körs. I uppstartsrutinen laddas alla från HMI-modulen inställningsbara värden, parametrar och mappar, in från permanentlagringen i flashminnet. Värdena placeras i minnesblock som används när systemet exekveras.

4.2.6 Loggrutin

Detta subsystem skickar loggdata till HMI-modulen, denna exekveras på tidsbas med en periodtid på 2 sekunder.

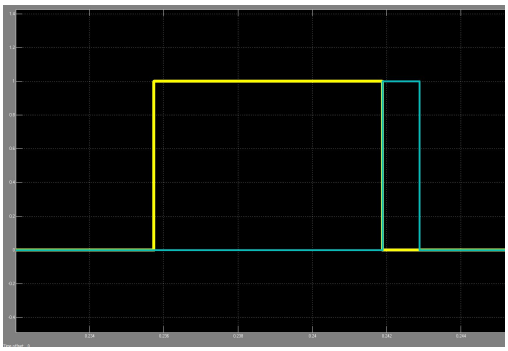


Figur 4.13: Loggrutin

De värden som när rutinen exekveras ligger i minnesblocken(Figur4.13(1)) skickas av UART-blocket(4.13(4)) till HMI-modulen. UART-blocket hämtar och skickar en textsträng från ett datalagringsblock(4.13(3)), textbehandlingsblocket(4.13(2)) skriver en textsträng, innehållande loggdatan, på ett för komunikation mellan modulerna förbestämt format till datalagringen. I figuren ses även det block(5) som används för inställning av UART-enheten på mikrokontrollern.

4.2.7 Simulering

För verifikation av styralgoritmernas funktion under utvecklingen har flera simuleringar i Simulink gjorts. Vid simulering kan interna och externa signaler mätas i modellen. De kan sedan jämföras med mätningar på det fysiska systemet när mikrokontrollern kör den kod som genererats från modellen. Som kan ses Figur 4.14 och Figur 4.15 stämmer utsignalarena från simulationen och mikrokontrollern bra överens med varandra. Figur 4.14 visar den simulerade signalen och i Figur 4.15 ses den verkliga signalen från styrsystemet mätt med oscilloskop.



Figur 4.14: Utsingalerna till H-bryggan vid simulation av Simulinkmodellen.



Figur 4.15: Utsignalerna till H-bryggan vid mätningar på verkliga systemet.

Mätningarna är gjorda på de pulser som styr spänningen över spridaren. När signal 1(gul) är hög är spänningen över spridaren +12 V och när signal 2(blå) är hög är spänningen över spridaren -12 V. Signalerna styr en H-brygga vars konstruktion och funktion beskrivs i avsnitt 4.3.8.

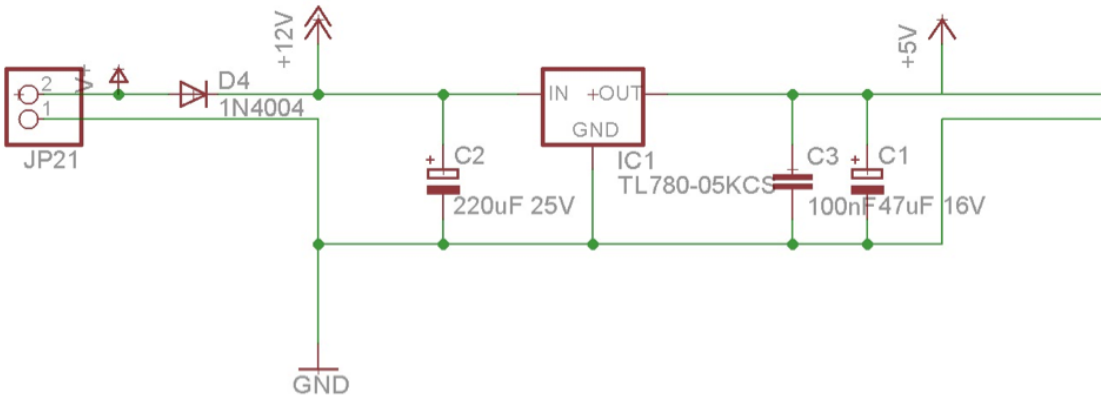
Till simulinkmodellen har ett MATLAB-program för generering av simulationssignaler tagits fram. Programmet genererar tidsserier för simulering av vev- och kamaxelsensorerna vid linjär acceleration från ett startvarvtal till ett slutvarvtal över en simuleringstid. Varvtal, tid och andra systemparametrar kan ställas i m-filen.

4.3 Elektronik

Här presenteras de kretsar som byggts för att hantera signalerna från de olika sensorerna samt drivelektronik för tändning och insprutning.

4.3.1 Spänningsförsörjning

Styrsystemet skall drivas av bilens batteri, detta levererar en spänning på 12 V. Både STM32F4 Discovery och RPi:n drivs med 5 V. För att reglera spänningen från 12 V till 5 V används en spänningsregulator(TL780-05KCS).



Figur 4.16: En spänningsregulator används för att försörja styrsystemet med rätt spänning

Kretsen för spänningsreglering kan ses i Figur 4.16. Spänningsregulatorn tar in en spänning på "IN" som kan variera mellan 7 V och 25 V och ger konstant 5 V på "OUT". Kondensatorerna C1, C2 och C3 är till för att kompensera variationer i strömförbrukningen och minska högfrekventa störningar. De är valda enligt tillverkarens rekommendationer.

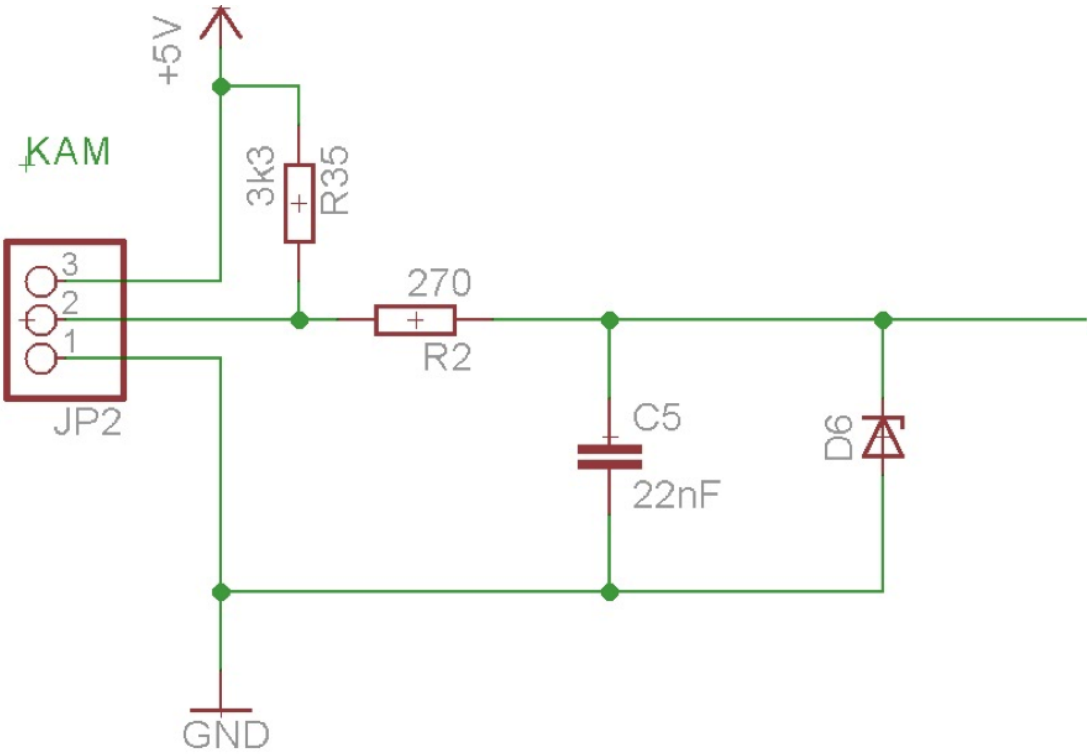
JP21 är anslutningskontakter för batteriet och D4 är en diod som skyddar systemet om användaren skulle koppla polerna fel. Ett spänningsfall på 0,7 V faller över dioden så spänningen vid 12V-pilen är egentligen 11,3 V men detta påverkar inte funktionen. Övriga kretsar i rapporten som har pilen 12 V eller 5 V är kopplade till samma punkt i kretskortet.

Styrsystemet använder två spänningsregulatorer, en för ECU-modulen och en för HMI-modulen, detta eftersom att de genererar mycket värme. Effekten blir differensen mellan ingående effekt och utgående effekt och HMI-modulen kommer kunna dra strömmar upp till 2 A. Värmegenereringen kommer således ha en effekt på(12 V - 5 V) * 2 A = 14 W.

Med en omgivningstemperatur på 25°C, bör spänningsregulatorn ha en kylare med en termisk resistans på 6,7 °C/W[11]. Kylaren EV-T220-64E har använts under utvecklingen men tanken är att när låda till styrsystemet byggs skall regulatorerna kylas av den.

4.3.2 Signalanpassning vevaxelsensor och kamaxelsensor

Vev- och kamaxelsensorerna kan egentligen kopplas direkt till mikrokontrollern men då Chalmers Vera Team har haft problem med störningar i signalen, speciellt från tändspolen, har signalen filtrerats.



Figur 4.17: Signalanpassning för kam- och vev-axelsensor

Kretsen illusteras i Figur 4.17:

- Sensorn kopplas in på JP2
- R35 är ett motstånd som måste vara anslutet för att sensorn skall fungera, se avsnitt 3.3.1.
- R2 tillsammans med C5 bildar ett lågpassfilter, se ekvation (4.8) och (4.9).
- D6 är en Zenerdiod som skyddar mot spänningar över 5,1 V.

Vid maxvarvtal, ca 7000 RPM, kommer pulserna ligga på:

$$f = \frac{7000}{60} * 24 = 2{,}8kHz \tag{4.8}$$

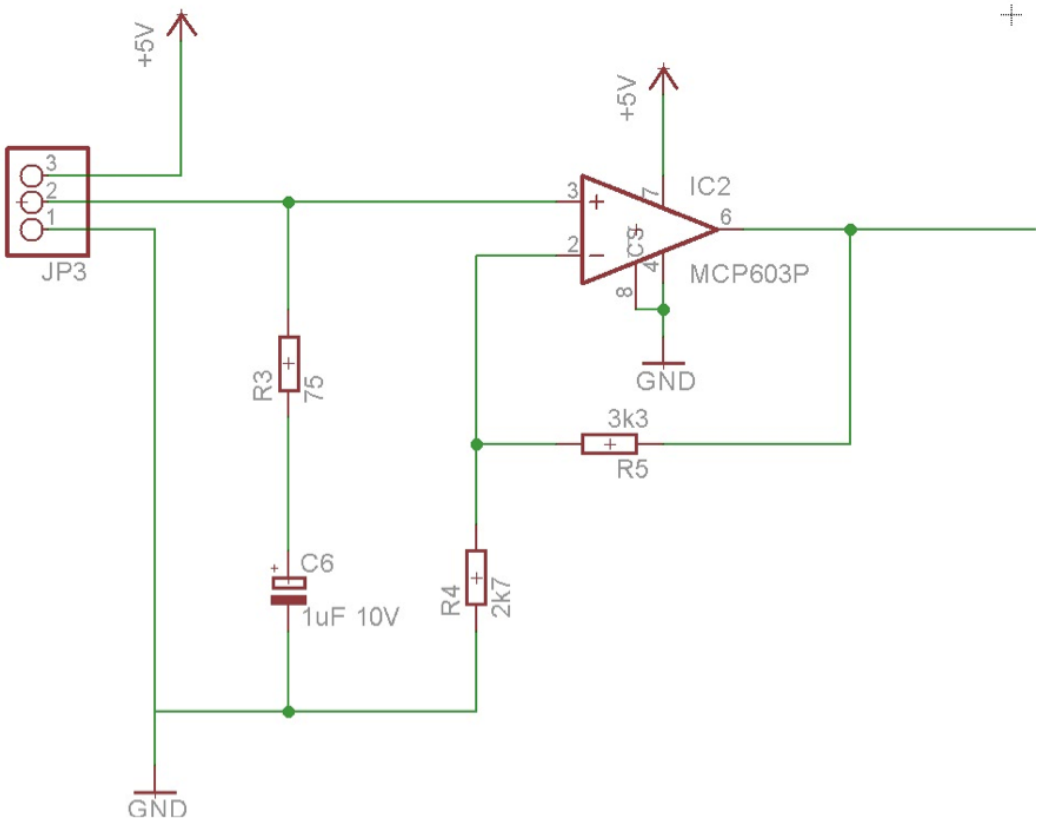
Filtrets brytfrekvens väljs till att vara åtta gånger högre än pulsfrekvensen för att inte förvränga signalen för mycket[12]. R2 väljs till 270 Ω, C5 bör därför vara på:

$$\frac{1}{2 * \pi * R2 * 8 * f} = 26{,}3nF \tag{4.9}$$

C5 väljs till 22 nF för att ta närmsta undre standardvärde.

4.3.3 Signalanpassning temperatursensor

Signalen från temperatursensorn är en spänning som förändras enligt 10 $mV/°C$ och med uppkopplingen i Figur 4.18 kan temperaturer från 0 $°C$ till 150 $°C$ mätas. Spänningen från sensorn kommer således variera mellan 0 V och 1,5 V.



Figur 4.18: Signalanpassning för temperatursensorer

Anpassning till AD-omvandlaren görs med en icke-inverterad förstärkare som är uppbyggd av en operationsförstärkare, IC2, och två motstånd, R4 och R5.

Förstärkningen för icke-inverterande förstärkare är:

$$\frac{V_{ut}}{V_{in}} = 1 + \frac{R5}{R4} \Leftrightarrow R5 = 1{,}2 * R4 \tag{4.10}$$

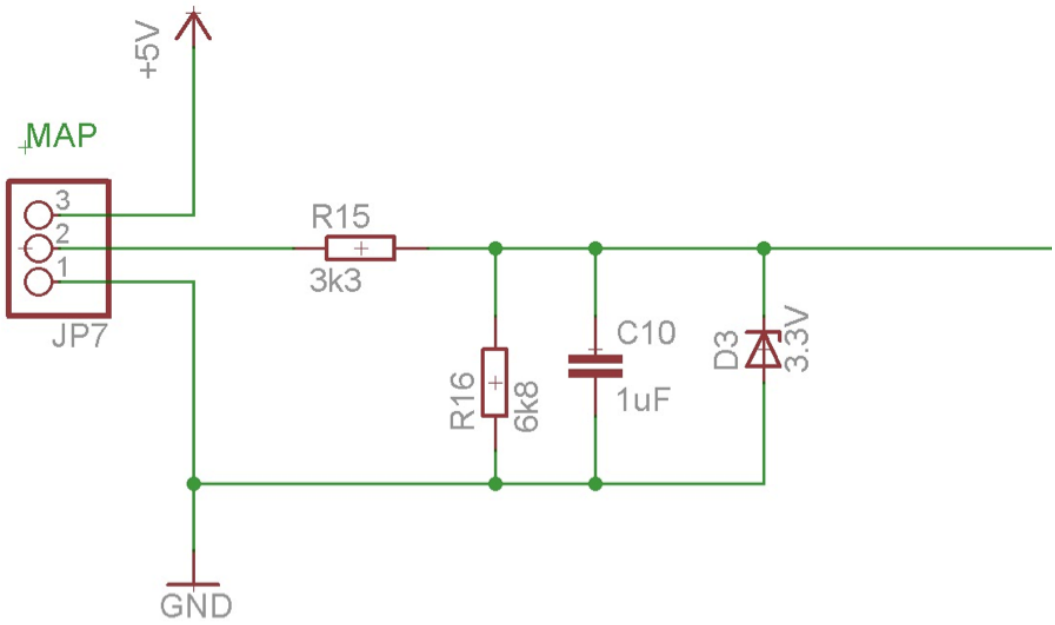
R4 valdes till 2,7 $k\Omega$ vilket ger R5 = 3,24 $k\Omega$. R5 valdes dock till närmsta standardvärde, alltså 3,3 $k\Omega$. Förstärkningen blir då 1 + 3,3/2,7 = 2.22... = 2.2.

R3 och C5 bildar tillsammans ett filter som tillverkaren för sensorn rekommenderar när sensorn sitter i en miljö med mycket elektromagnetiska störningar.

I denna krets fungerar operationsförstärkaren även som en spänningsbegränsning då utsignalen inte kan överstiga 5 V, detta är en egenskap hos operationsförstärkare. Som tidigare nämnts tolererar mikrokontrollern spänningar på 5 V.

I mjukvaran behöver temperatursensorernas signaler multipliceras med en faktor 45 för att justera värdet från AD-omvandlaren till rätt enhet($°C$). Detta då 150 $°C$/3.3 V = 45 $°C/V$.

4.3.4 Signalanpassning MAP-sensor



Figur 4.19: Signalanpassning för MAP-sensor

MAP-sensorn kan mäta mellan 20 kPa till 115 kPa och ger då 0,4 V respektive 4,65 V. Mellan de punkterna är utsignalen linjär. R15 och R16, i Figur 4.19, bildar en spänningsdelare som anpassar signalen till AD-omvandlaren, vid 115 kPa blir spänningen in till mikrokontrollern 3,13 V istället för 4,65 V(ekvation 4.11).

Spänningsdelning:

$$V_{R16} = \frac{R16}{R16 + R15} * 4{,}65 = 3{,}13V \tag{4.11}$$

C10 tillsammans med R15 bildar ett lågpassfilter och D3 är en Zenerdiod som skyddar mot överspänningar.

För att ge rätt enhet efter AD-omvandlaren i mjukvaran krävs i detta fall både en proportionell faktor och en additiv term. Detta kommer ur räta linjens ekvation(ekvation 4.12).

$$p = ku + m \tag{4.12}$$

Spänningsdelningens dämpning, a=6,8/(6,8+3,3)=0,67 och den proportionella faktorn

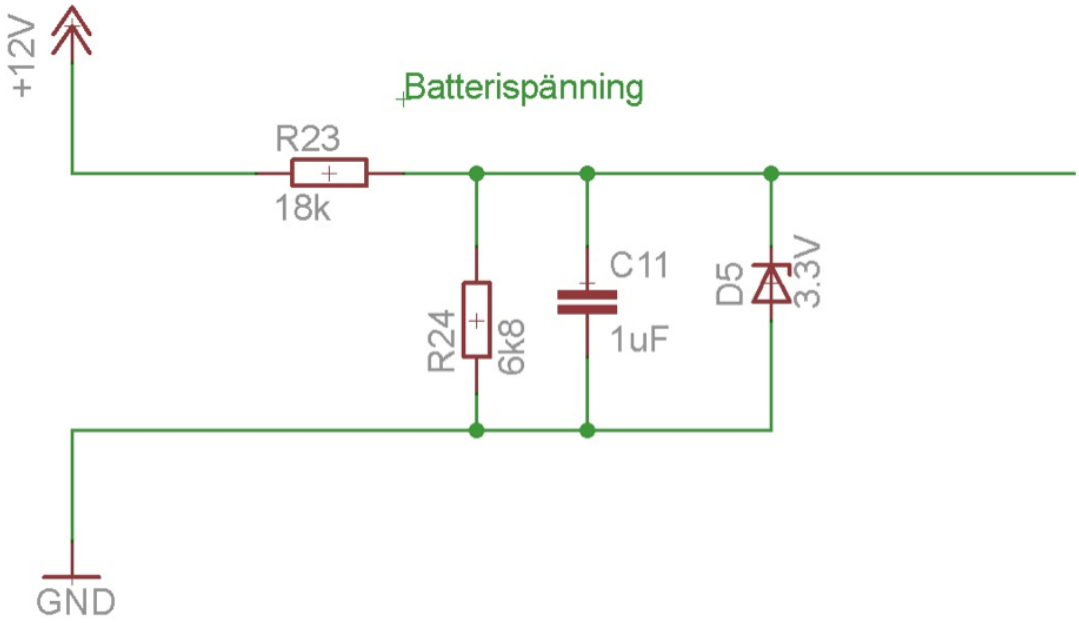
$$k = \frac{115kPa - 20kPa}{a * (4{,}65V - 0{,}4V)} = 33{,}2 \tag{4.13}$$

vilket gör att den additiva faktorn

$$m = 115kPa - a * 4{,}64 * k = 11{,}06. \tag{4.14}$$

4.3.5 Mätning av batterispänning

Anpassningen för att mäta batterispänningen görs på samma sätt som signalanpassningen för MAP-sensorn, med en spänningsdelare, se avsnitt 4.3.4. Endast värdena på motstånden skiljer sig, se Figur 4.20 .



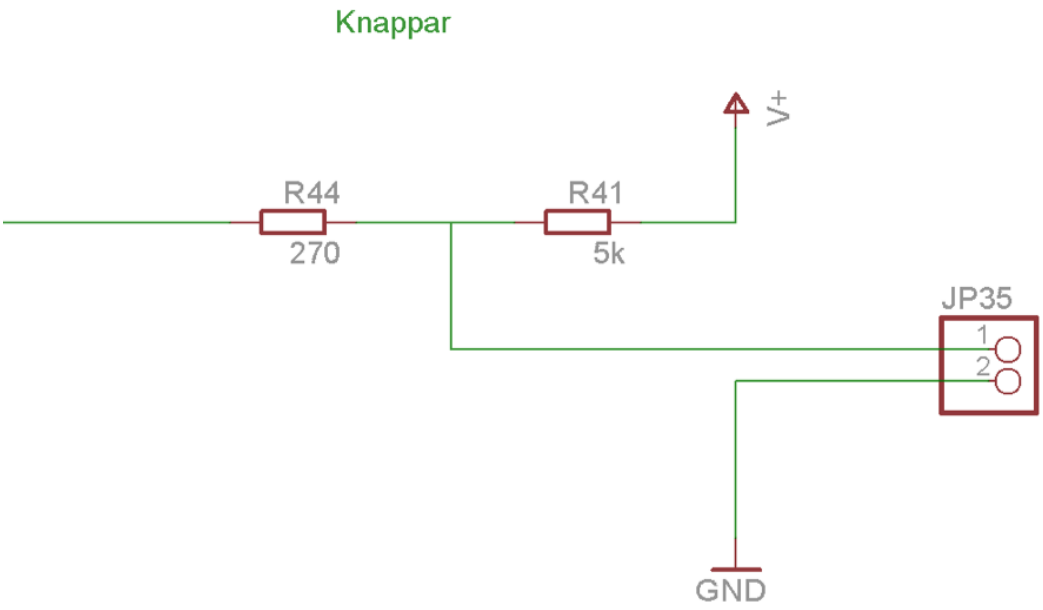
Figur 4.20: Krets för att mäta batterispänning

Vid 12 V blir spänningen till mikrokontrollern 3,29 V enligt samma resonemang som ekvation 4.11.

För att skala värdet från AD-omvandlaren behöver detta multipliceras med 12 V / 3,29 V = 3,64 samt 0,7 adderas för att få ut batterispänningen i Volt i mjukvaran. Additionen görs för att kompensera för spänningsfallet över skyddsdioden på ingången(se avsnitt 4.3.1).

4.3.6 Knappar

Knappar har implementerats som i Figur 4.21.



Figur 4.21: Illustrerar hur knappar ansluts i styrsystemet

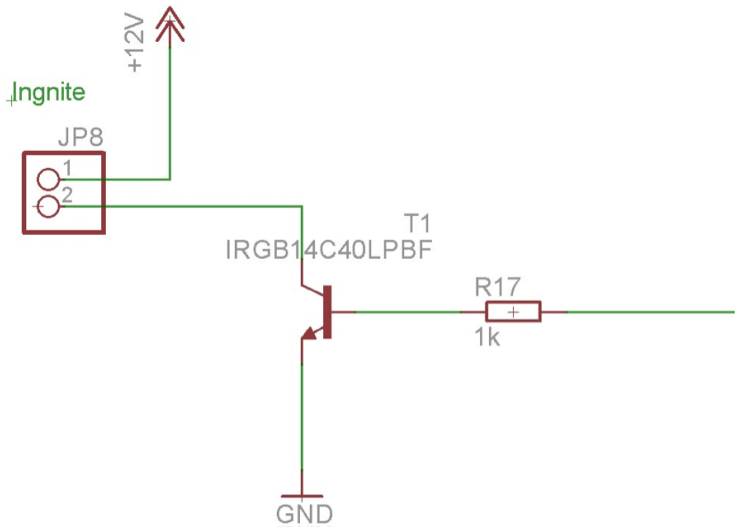
Knappen kopplas in på JP35. R41 är ett motstånd som drar spänningen mot 5 V när knappen ej är nedtryckt, insignalen är då hög. När knappen trycks in blir insignalen 0 V, alltså låg. *Debounce* implementeras mjukvarumässigt och i Waijung finns färdiga block för detta.

R44 är ett strömbegränsande motstånd i syfte att skydda mot fel. Maxström som mikrokontrollern klarar av är 25 mA och R44 måste därför minst vara 200 Ω(ekvation 4.15), 270 Ω valdes för att ha en säkerhetsmarginal.

$$\frac{5V}{25mA} = 200\Omega \tag{4.15}$$

4.3.7 Drivelektronik tändning

Kretsen som valdes heter IRGB14C40LPBF(se Figur 4.22) och är en IGBT som är anpassad för tändning med tändspole på mindre motorer. Den har inbyggda skyddskretsar mot överspänning från tändspolen. Vid JP8 kopplas tändspolen in och R17 är ett strömbegränsande motstånd, $1k\Omega$ rekommenderades av tillverkaren.



Figur 4.22: Drivkrets för tändning. IGBT

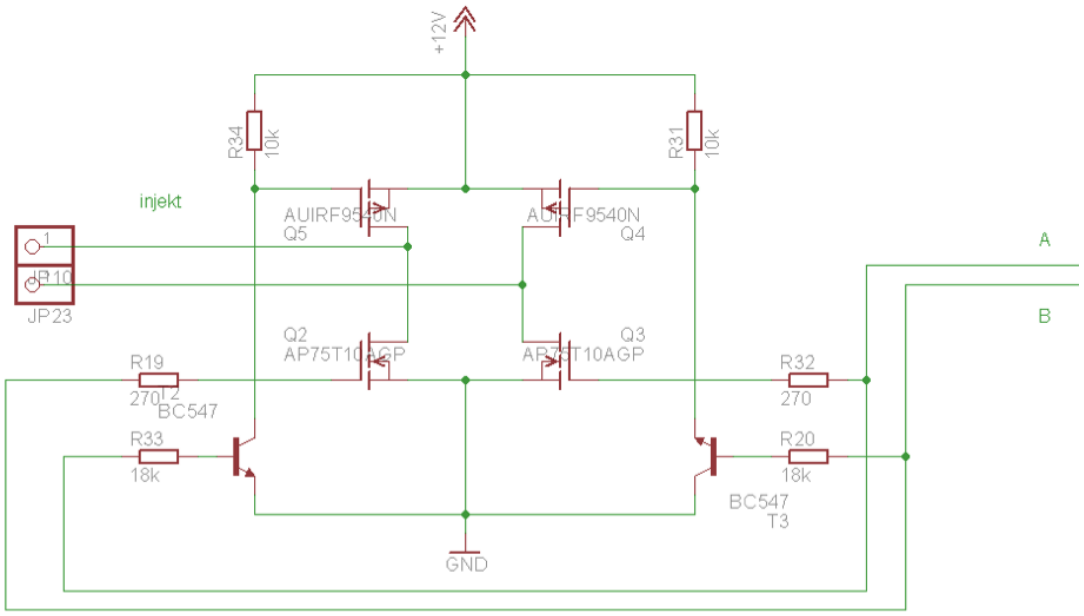
4.3.8 Drivelektronik insprutning

Drivkretsen bygger på en H-brygga som är uppbyggd av två nMOSFET och två pMOSFET, se Figur 4.23.

- nMOSFET (Q2 och Q3): AP75T10AGP, 105 V, 65 A
- pMOSFET (Q4 och Q5): AUIRF4905, -55 V, -74 A.

MOSFET:arna är anpassade för att kunna drivas av låga spänningar. Gränsspänningen för när transistorerna stryps(eng. Threshold Voltage) är 1 V respektive -2 V vilket gör att de kan drivas direkt från mikrokontrollern. De har även inbyggda skyddsdioder.

Q4 och Q5 är byggda för att hantera negativa spänningar och strömmar och kopplas således omvänt mot Q2 och Q3 med sin *drain* mot jord istället för *source*.



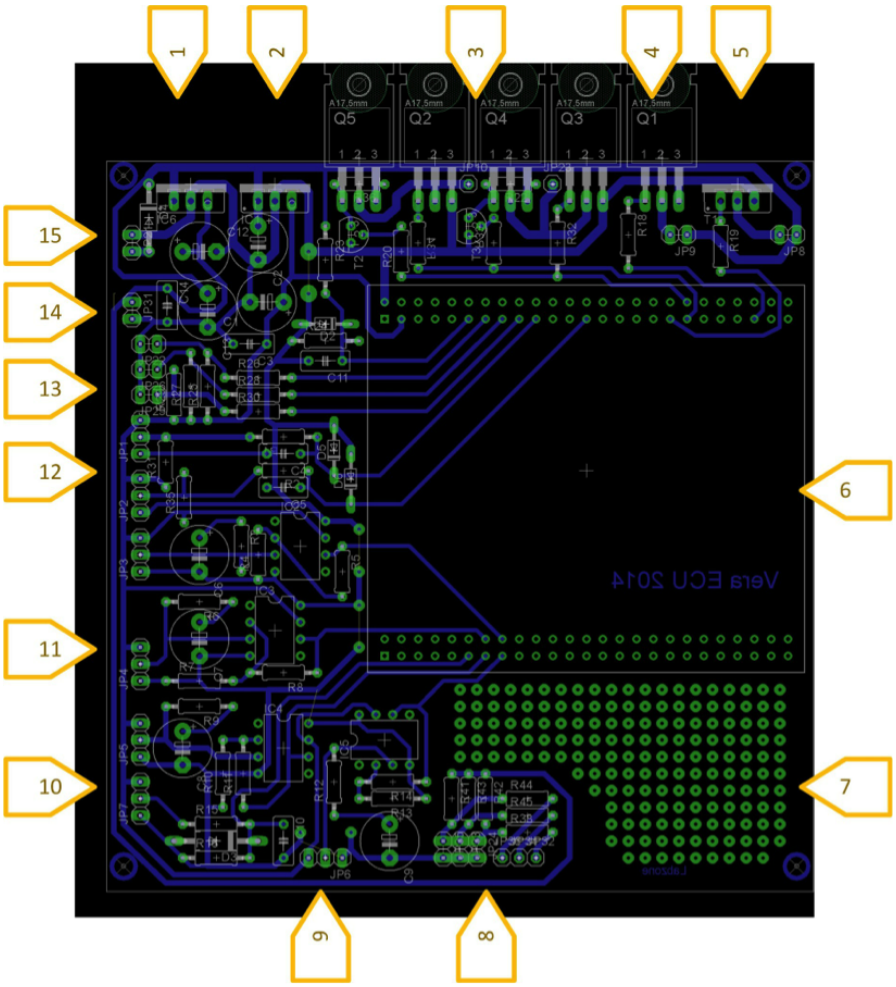
Figur 4.23: Kretsschema på styrsystemets h-brygga som möjliggör polvändning av strömmen till injektorn.

Injektorn/spridaren kopplas in på JP23, A och B kopplas till mikrokontrollern.

När A är hög och B är låg blir spänningen över *gate* och *source* på Q3 3,3 V och då leder den. Samtidigt går det en ström genom transistorn T2 vilket gör att spänningen över R34 närmar sig 12 V. Spänningen över Q5:s *gate* och *source* blir således -12 V, då leder även Q5. Q4 och Q2 är i detta sammanhang strypta och spänningen över injektorn blir +12 V. För att få motsatt polaritet sätts först A till låg, därefter sätts B hög. Då sker omvänt från ovanstående och spänningen över injektorn blir -12 V.

4.3.9 Kretskort

Kretskortet är designat för hålmonterade komponenter och har endast ett kopparlager. STM32F4Discovery:n sitter fast i en socket och kan således lätt plockas bort. Den elektronik som går på 12 V(drivkretsar för tändning och insprutning) sitter i övre delen av kortet där kopparbanorna är bredare.



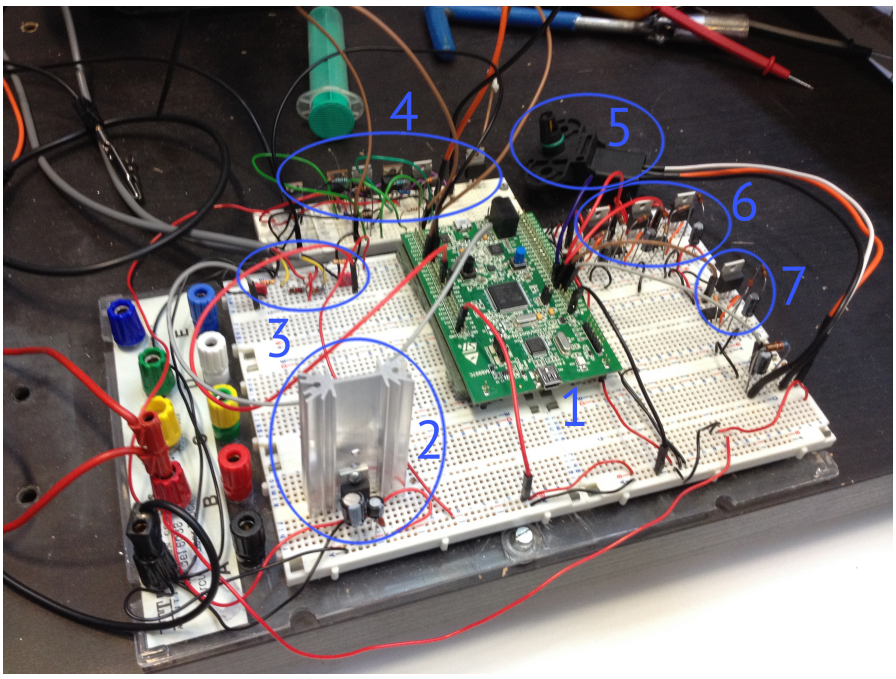
Figur 4.24: Ritning på kretskort som gjorts för styrsystemet

Beskrivning av kortet, Figur 4.24 :

1. Spänningsregulator TL780-05KCS för HMI.
2. Spänningsregulator TL780-05KCS för ECU.
3. Drivkrets för spridare, H-brygga.
4. Drivkrets för spridare som beskrivs i kapitel 3.4.2 .
5. Drivkrets för tändning.
6. Socket för STM32F4Discovery.
7. Labzone, för att enkelt kunna montera nya komponenter för vidare utveckling.
8. Anslutning för knapp, HMI.

9. Anslutning för temperatursensor, x1.
10. Anslutning för MAP-sensor.
11. Anslutning för temperatursensor, x3.
12. Anslutning för vevsensor och kamsensor.
13. Anslutning för knappar, ECU.
14. 5 V och GND för RPi.
15. Anslutning för batteriet, 12V.

Kretskortet har inte tillverkats utan kretsarna har verifierats på ett kopplingsdäck(Figur 4.25).



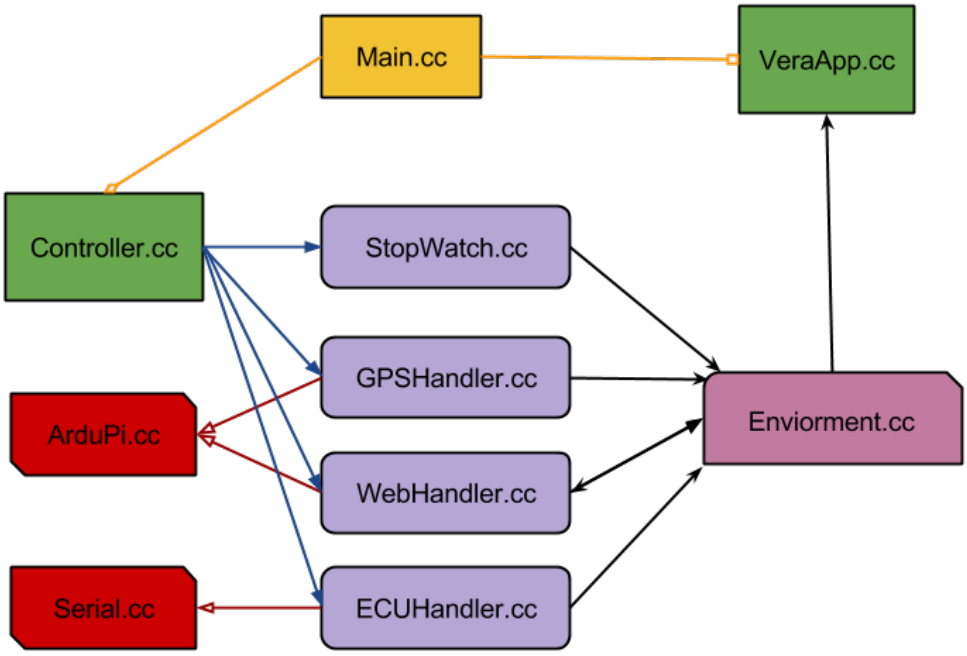
Figur 4.25: Uppkoppling på kopplingsdäck. STM32F4-Discovery(1), Spänningsregulator ECU(2), signalanpassning för kam- och vevaxelsensorer(3), drivkretsar för tändning och spridare(4), MAP-sensor(5), motortemperatursensorer(6), lufttemperatursensor(7).

4.4 Programvara för HMI

I detta avsnitt presenteras HMI-programvarans struktur och resultatet av den kod som utvecklats.

4.4.1 Grundläggande arkitektur

Arkitekuren är uppbyggd av en kontrollklass, en miljöklass och en GUI-klass. Kontrollerklassen har hand om alla klasser som tar emot data från olika källor, så kallade Handlerklasser, och hanterar även när föraren trycker på knapparna för stoppuret. ”Handlerklasserna kan först behandla datan internt, exempelvis lägga in den i databasen, för att sen uppdatera miljöklassen. Alla Handler-klasser körs även i separata trådar. När miljöklassen känner av att ny data finns uppdaterar den användargränssnittet automatiskt. Miljöklassen håller även senaste värderna, så om en Handler-klass behöver data ifrån en annan kan dessa fås via miljöklassen. Detta sker bland annat i WebHandler där GPSpositionen behövs. Hela arkitekturen illustreras i Figur 4.26



Figur 4.26: HMI Arkitektur

Nedan följer en lista och en kort beskrivning av de viktigaste klasserna:

- StopWatch.cc - Ansvarar för klockan och varvtiderna genom att ta emot styrsignaler ifrån Controller om när klockan t.ex. ska sättas på eller stängas av. Den uppdaterar kontinuerligt miljöklassen så att GUI:t uppdaterar sig med den aktuella tiden.

- GPSHandler.cc - Ansvarar för hitta och uppdatera GPS positionen genom att kontinuerligt skriva AT-kommandot till GPS enheten och sedan extrahera positionen ifrån svaret.
- WebHandler.cc - Ansvarar för 3G-enheten och att uppdatera livedata genom att hämta den relevanta datan ifrån miljöklassen och sedan skicka upp detta via 3G-enheten med hjälp av en HTTP-request.
- ECUHandler.cc - Ansvarar för kommunikationen och loggningen ifrån ECU:n genom att kontinuerligt avläsa data ifrån den seriella porten, extrahera datan, lägga in den i databasen och sedan uppdatera miljöklassen så att GUI:t uppdaterar sig.
- Enviorment.cc - Hanterar uppdateringen av GUI:t och håller värden på globala variabler.
- VeraApp.cc - Själva GUI:t som innehåller metoder för att uppdatera alla grafiska element.
- ArduPi.cc - Bibliotek för kopplingarna till GPS/3G-enheten.
- Serial.cc - Bibliotek för den seriella USB enheten.

4.4.2 Gränssnitt till föraren

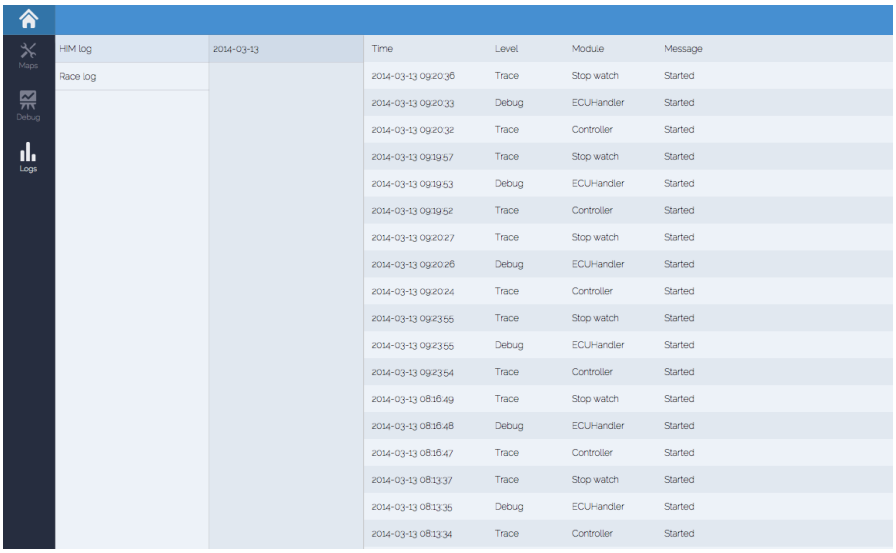
Utifrån resultatet av intervjun med föraren konstruerades ett användargränssnitt i Glade där tid och hastighet var de centrala delarna. För att uppnå maximal läsbarhet och kontrast valdes en svart bakgrund med vit text. Två statussymboler för att indikera om enheten är ansluten till GPS och ECU-modulen placeras uppe i högra hörnet. För tydlighet blir texten röd om ingen anslutning finns och grön när enheterna är anslutna. Uppe till vänster finns ett fält där felmeddelanden kan skrivas ut. Detta kan gälla fel i både HMI-modulen och ECU-modulen. Längst ner, från vänster, presenteras medelhastighet, varvtid, motorns varvtal och antal varv i racet i tur och ordning(Figur 4.27).



Figur 4.27: Gränssnitt mot föraren

4.4.3 Loggning

ECU-modulen skickar loggdata med ett jämt intervall. Dessa tas emot i ECUHandler där dels GUI:t uppdateras och dels läggs data in i databasen. För att stoppa in datan i databasen används standard c++ bindningar till mysql. Även loggar ifrån HMI-programvaran sparas ner till databasen och visas på webbsidan(Figur 4.28). Motordata som lagrats unders race kan exporteras ut till csv-filer för att analyseras i annan programvara.



Figur 4.28: Webbsida för visning av HMI loggar

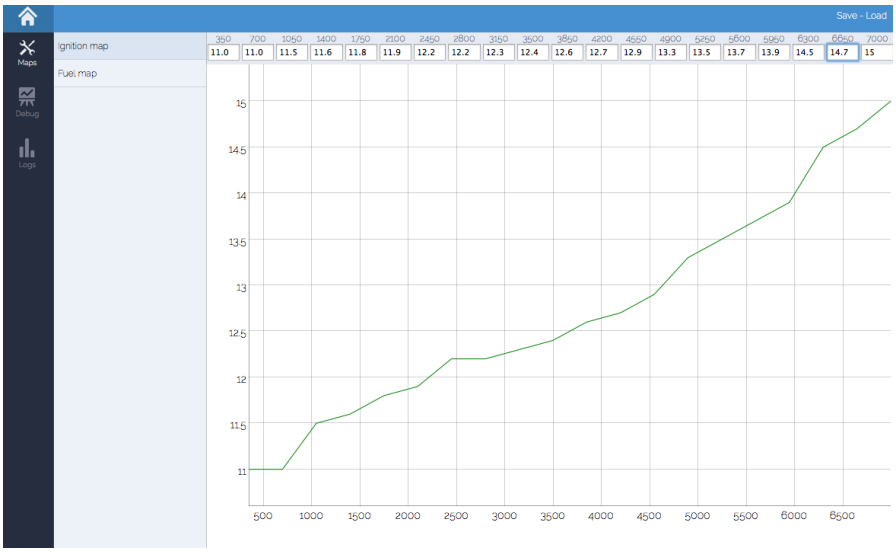
Meddelandena som ECU-modulen skickar och HMI-modulen tar emot ser ut som följande:

!LOG:data1+data2+data3+data3&

All kommunikation startar med ett ! och avslutas med ett &.

4.4.4 Programmering av ECU mappar

Programmeringen av mapparna sker, som tidigare nämnts, via en webbsida, se figur 4.29. Data hämtas och sparas på ECU-modulen via ett antal steg. Först görs en AJAX-request till en PHP-sida som bearbetar datan och kör ett pythonscript med argument. Pythonscripten är mycket simpla och skriver och läser endast på den seriella porten. Resultatet skickas från pythonscripten till PHP-sidan och skickas sedan vidare till webbsidan.



Figur 4.29: Webbsida för programmering av mappar

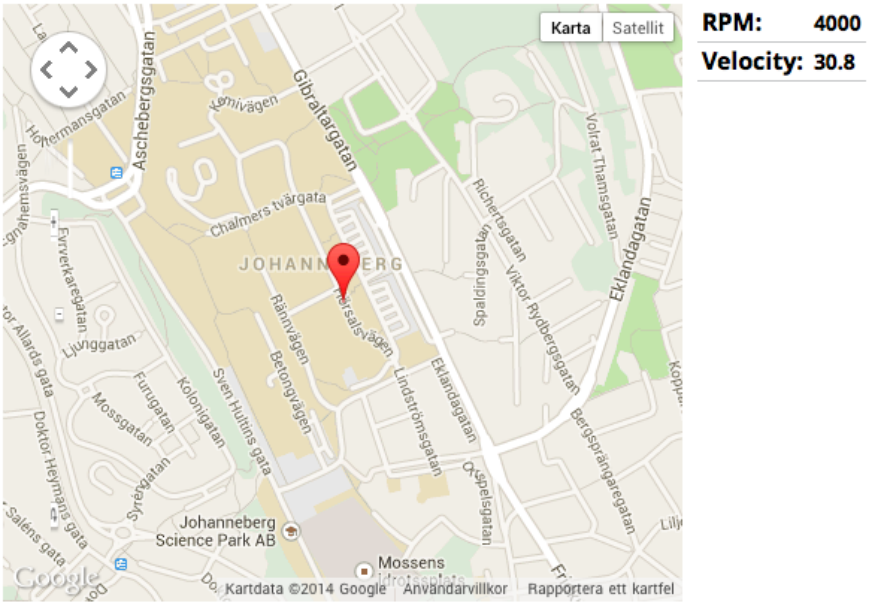
Dataprotokollet för det som skickas ser ut som följer:

HMI get	@GET:mapid&	ex @GET:1&
ECU svar	#OK:värde1+värde2+värdeN&	ex #OK:1+2+3+4+5+6+7&
HMI set	@SET:mapid:värde1+värde2+värdeN&	

I ECU-modulen används FLASH-minne för att simulera EEPROM-minne.

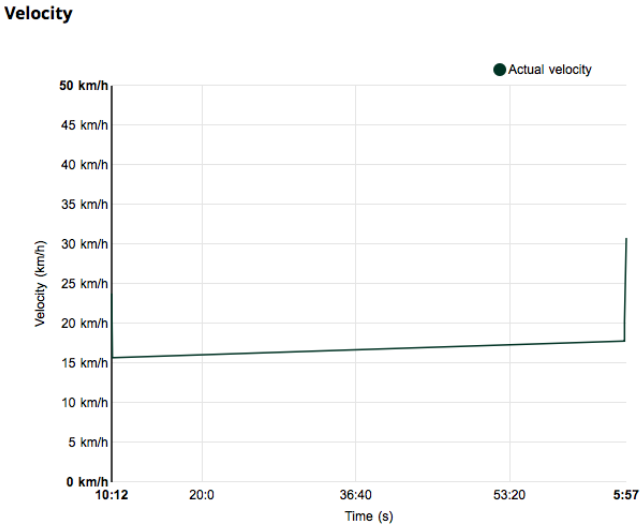
4.4.5 Livedata

Uppdateringen av livedata sker i flera steg. RPi:n kopplar upp sig mot 3g-nätet med hjälp av 3g-modulen via AT-kommandon. Sedan frågas respektive Handler-klasser om värden som ska laddas upp. Uppkoppling sker mot servern och ett http GET-request med de värden som ska läggas in i databasen skickas. Requestet behandlas i en PHP-fil som lägger in värdena i en MySQL-databas.



Figur 4.30: Karta på vilken Veras position visas

Webbsidan uppdaterar sig själv med jämna mellanrum på intervall av 5-10 sekunder och hämtar då de värden som senast lades in i databasen. Det är alltså inte alltid de senaste värdena som visas på hemsidan, men intervallet har bestämts så att värdena som visas ska vara tillräckligt rättvisande för att inte ge en felaktig bild av hur racet går. På sidan finns en karta(Figur 4.30) på vilken Veras aktuella position visas och en graf(Figur 4.31) där Veras hastighet över tid presenteras.



Figur 4.31: Graf för att visa hastighetsförändringar

5. Diskussion

Här presenterar diskussioner och slutsatser kring resultatet och de resonemang som har lett fram till detta. Förslag på framtida utvecklingsmöjligheter presenteras också.

5.1 Genomförande

Här diskuteras resonemangen kring val och lösningar på projektproblemen.

5.1.1 Val av programmeringsmönster

Initialt valdes ett programmeringsmönster där användargränssnittet hämtade data ifrån en miljöklass med en viss uppdateringsfrekvens. Detta gjordes på grund av osäkerheten kring den beräkningskraft det skulle ta att uppdatera användargränssnittet. Mottagning och hantering av loggdata ifrån ECU-modulen måste prioriteras så att ingen värdefull information försvinner på grund av fulla mottagningsbuffrar. Med en valbar uppdateringsfrekvens kunde beräkningskraft för att uppdatera användargränssnittet regleras manuellt i koden. Vid implementationen insågs att detta inte var något problem och då gjordes en mindre refaktorering så att uppdateringen i slutversionen utförs direkt när ett nytt värde finns tillgängligt.

5.1.2 Implementation av antal varv och varvtid

I förra årets projekt valdes att implementera varvigenkänning med en algoritm som använde GPS:en för att avgöra när Vera passerade startlinjen. Vi misstänkte att det skulle bli svårigheter med att få detta exakt och såg även problem med att tävlingarna sker på olika geografiska platser då positionen för start och mål ändras. Eftersom föraren var tydlig med att noggranna varvtider var något som var mycket viktigt bestämdes istället att göra det så enkelt som möjligt och därför valdes en knapp. Denna lösning har andra nackdelar, fel som att föraren trycker två gånger kan uppstå men vi anser ändå att detta är en stor förbättring jämfört med innan eftersom föraren nu endast behöver komma ihåg att trycka på en knapp vid startlinjen istället för att behöva komma ihåg exakt antal varv i huvudet hela tiden. Dessutom får föraren exakta varvtider vilket inte funnits innan.

Här finns stora möjligheter för vidare arbete. För att eliminera interaktionen med föraren borde systemet kunna sköta detta automatiskt. Antingen bygga och testa en avancerad GPS-algoritm eller utveckla systemet med någon sändare och mottagare vid mållinjen.

5.1.3 Valet av Raspberry Pi och Raspbian

Att använda en Raspberry Pi och Raspbian är kanske inte det bästa valet. De valdes för att snabbt kunna börja utveckla och för att de uppfyllde kraven som ställts på hård- och mjukvaran. Här finns stora möjligheter till förbättringar genom att antigen byta ut

Raspbian mot ett operativsystem som är mer optimerat för användningsområdet eller skala bort onödig funktionalitet ifrån det nuvarande operativsystemet, få av de funktioner som finns använts. Storleken på Raspbian versionen som används är ungefär 800 MB och exempelvis finns Tiny Core Linux for Raspberry Pi på 32MB. Vid ett byte eller nerskalning skulle stora förbättringar i uppstartstider och minskad resursanväning kunna uppnås. Ju mindre resurser som använts, desto mindre kommer processorn jobba vilket medför mindre strömförbrukning.

Det kan även finnas fördelar att på lång sikt byta ut hela RPi:n till någon annan typ av enkortsdator. Det finns många typer av enkortsdatorer och nu i efterhand inses att en med exempelvis inbyggt Wi-Fi och två stycken UART-moduler skulle passa bättre. Det skulle helt ta bort enheterna som är inkopplade via USB, vilket skulle minska storleken och öka enhetligheten.

5.1.4 Valet att inte använda korskompilering

För att snabbt komma igång och ge mer frihet i utvecklandet valdes alternativet med att korskompilera bort. Detta alternativ innebär att man utvecklar på en typ av plattform och sedan kompilerar sin kod mot en annan plattform. Utvecklingen hade då kunnat ske på Windows eller OS X, kompileras på kraftfulla datorer för att sedan installeras på RPi:n, som kör Linux. Fördelarna med detta hade varit att kompileringen hade gått snabbare och utvecklingen skulle kunna ske offline. Nackdelarna hade varit en komplicerad uppstartsprocess och minskad flexibilitet med att använda externa bibliotek vid utvecklingen av kod. Detta hade varit ett bra alternativ om utvecklingen skulle pågå länge och kompileringen skulle vara väldigt resurskrävande, vilket inte var fallet i detta projekt.

5.1.5 Mjukvaruutveckling i Simulink

Mjukvaruutvecklingen i Simulink var till en början mycket omständig, inlärningstiden blev lång. Detta berodde till stor del på bristfällig dokumentation till Waijung, dokumentationen består i stort sett av ett enkelt exempel per block, inga generella anvisningar till hur blocken skall användas och dess begränsningar finns. Detta i kombination med att det i Simulink finns en uppsjö av olika sätt att ställa upp samma stycke av en modell på innebar att mycket experimenterande behövde utföras för att finna en metod som fungerade, detta för i princip varje block i stort sett i varje ny situation. Vid fortsatt arbete på denna modell rekommenderas därför att de metoder som använts i modellen följs. I flera fall kan använd metod verka omständig och det kan tyckas finnas uppenbara enklare sätt att göra på, dessa har sannolikt testats och visat sig inte fungera.

Eftersom mycket experimenterande krävdes så hade traditionell mjukvaruutvecklingsmetod varit en effektivare metod för att programmera mikrokontrollern. Ser man inga andra värden i att ha en Simulink-modell och saknar tidigare kunskap om Simulink så kan man med fördel välja att programmera på traditionellt vis.

5.1.6 Tankar kring det som inte täcktes in av projektet

Några av de kraven som gavs har inte uppfyllts då tidsramen för projektet visade sig vara för kort. Som nämnts i kapitlet Avgränsningar har fokus legat på att bygga en bra grund som sedan kan vidareutvecklas. Många av de kraven som är kvar handlar inte om systemets uppbyggnad eller konstruktion utan är olika funktioner som behöver implementeras.

Våra tankar kring några av de funktioner som inte implementerats:

- Felkoder: I HMI-modulen är detta förberett så implementationen borde vara relativt simpel. Eventuella felkoder som kan skickas är om värdena från sensorerna hamnar utanför givna gränsvärden, batterispänningen sjunker eller om vevaxelsensorn missar avkänningpunkter.
- Start och stopprutin: Utveckla en rutin som hanterar hur motorn startas och stoppas. Tankar finns på att göra detta i Stateflow(annat tillägg till MATLAB) då det är många logiska samband som avgör när motorn skall starta och stoppa. Många av de krav som inte är uppfyllda handlar om hur motorn startas och stoppas.
- Bygga låda: Styrsystemet måste placeras i en vattentät låda. Viktigt är att den kan leda värme från spänningsregulatorerna, så metall är ett bra val. Annars behöver spänningsförsörjningen göras om.
- Kompensation för knack: Idéer har funnits på att göra detta med ett bandpassfilter och någon form av komparator. Enligt T-engineering är detta en omständig lösning. Bättre är att sampla signalen och fouriertransformera så att de rätta frekvenserna kan upptäckas.
- Kompensation för luftfuktighet: Efter en ofullständig litteraturstudie i ämnet har vi fått indikationer på att kompensation för luftfuktighet måste göras både för bränsleinsprutning och tändning. Detta då luftfuktigheten dels påverkar luftens andel syre, därför skulle mindre bränsle vid fuktig luft behövas, och dels för att flamfronten skulle få en långsammare utbredning vid fuktig luft, då behöver gnistan komma tidigare.

5.2 Utvecklingsmöjligheter

Nedan följer ett antal förslag på vidareutveckling av systemet.

5.2.1 Utökat webbgränssnitt

Dagens webbgränssnitt är långt ifrån färdigt, det är mer en mall för framtida utveckling. Här finns det möjligheter att t.ex. ha fler motorparametrar att ställa in och analysverktyg för loggdatan. Man kan implementera logik som visar avvikande värden, genomsnittliga tider eller andra analyser som man vill ha tillgängliga direkt efter ett race utan att behöva exportera datan till andra programvaror. Det är endast fantasin som sätter gränser.

5.2.2 Hastighetsmätning

Att inte använda GPS:en som hastighetsmätare skulle ge en mycket mer exakt hastighet med en mycket högre uppdateringsfrekvens. Även ökad felsäkerhet då förlorad GPS-mottagning, som systemet nu är uppbyggt, skulle slå ut hastighetsmätningen. Möjligheten att ansluta en fysisk hastighetsmätare finns bland annat via I2C, digital ingång eller analog ingång.

5.2.3 Mer sensorer och utökad info till föraren

Via GPS/3G kortet ges möjligheten att läsa analoga signaler på RPi:n. Dessa kan användas för att ansluta andra typer av sensorer som man kan logga eller för att ge föraren annan information. Backkamera har disskuterats då synfältet är begränsat och andra bilars position kan vara direkt avgörande för vissa beslut. För att öka förarens möjligheter skulle skärmen kunna implementeras med hjälp av en surfplatta med pekskärm som ansluts till HMI-modulen via Wi-Fi. Vad detta skulle kunna användas till har inte reflekterats så mycket över, utan bara att möjligheten finns.

5.2.4 Höghastighetsloggning

Många kommersiella styrsystem har en loggfunktion där data för alla in och utsignaler loggas med högfrekvens för att kunna upptäcka fel. För att implementera höghastighetsloggning bör man analysera om STM32F4 processorn har tillräckligt med beräkningskraft för att hinna med det, eller om en till mikroprocessor ska användas för att logga signalerna. Om data från höghastighetsloggningen skall fås via webbgränssnittet på HMI-modulen eller via ett separat program är andra frågor som behöver redas ut.

Höghastighetsloggningen kommer generera mycket data så möjligheten att implementera USB VCP borde undersökas mer eftersom bandbredden och enhetligheten skulle öka avsevärt.

5.2.5 Datainsamling

Detta system skiljer sig från tidigare system i att det kan spara racedata. Detta är inte begränsat till den data som nu valts att sparas. Om lambdasensor och sensor för förbränningskammartryck skulle kopplas till systemet så kunde även denna data sparas. Skulle man göra detta kunde man över tid bygga upp en stor databas med data från både omgivningsfaktorer, input till systemet, och motsvarande data från systemets svar, output. Denna data kunde analyseras för att se hur enskilda omgivningsfaktorer påverkar systemet och för att utveckla styralgorimerna. Exempelvis kunde luftfuktighetens eventuella påverkan på var trycktoppen i förbränningsrummet hamnar undersökas, något som vi sökt information om utan att hitta. Dessa sensorer kunde också användas för återkoppling till och reglering av systemet under race.

5.2.6 Avsaknad av block för att sätta icke-periodiska mjukvaruavbrott

Att ett block där möjligheten att sätta timermoduler att ge icke-periodiska mjukvaruavbrott saknas i blocksetten har inneburit att ett periodisk mjukvaruavbrott som hela tiden är aktivt har fått användas. Denna metod har begränsningar, ett periodiskt mjukvaruavbrott innebär att CPU:n periodiskt måste hoppa från det den arbetar med och utföra en avbrottrutin, i detta fall att öka ett räknarvärde och se om det är läge att utföra en annan uppgift. Detta är inte effektivt och det resulterar i att processorns möjlighet att göra dessa hopp samtidigt som annan kod skall behandlas begränsar vilken precision som kan uppnås i denna tidsmätning, därmed också vinkelprecisionen med vilken vi kan utföra uppgifter. Om möjlighet hade funnits att sätta icke-periodiska mjukvaruavbrott hade detta inte varit ett problem, en timermodul i mikrokontrollern hade arbetat i bakgrunden och dennas tidsupplösning hade avgjort med vilken precision uppgiften kunnat utföras.

Ett möjligt sätt att komma runt detta problem hade varit att göra ett eget block som implementerar icke-periodiska mjukvaruavbrott. Efterforskningar kring detta resulterade i slutsatsen att detta inte rymdes inom projektets tidsram. En möjlighet att effektivisera och förbättra systemet i framtiden skulle vara att göra ett sådant block. Innan det arbetet börjar rekommenderar vi att man tar en titt på Waijungs hemsida då blocksettet uppdateras kontinuerligt.

6. Slutsats

Projektets syfte var att utveckla att styrsystem som skulle ge Chalmers Vera Team bättre möjlighet att analysera motordata och som enkelt skulle vara utbyggbart vid införande av ny motorteknik.

Styrsystemets funktion har verifierats med hjälp av testriggen. Motorn startar och genom att ändra värden i mapparna förändras motorns beteende vid olika vartal. Inga mätningar för att verifiera att regleringen för luftryck och lufttemperatur har gjorts. Simululeringar i Simulink pekar dock på att regleringen fungerar som tänkt.

Motordata görs av systemet tillgänglig för analys, dels sparas data i en databas och dels så presenteras en del data på internet live under körning vilket möjliggör analys och övervakning av bilen under race.

Styrsystemet är också designat för att enkelt kunna vidareutvecklas. Simulinkmodellen möjligör enkel visareutveckling av ECU-modulens programvara. Simulinkmodellen har också förberetts för ny motorteknik som ligger nära i framtiden. HMI-modulen har många möjligheter att utvidgas med nya funktioner, exempelvis bättre analysverktyg i webbgränssnittet.

Även om vissa krav som presenterats i kravspecifikationen inte har uppfylts och styrsystemet inte genomgått en omfattande utvärderingsprocess anser vi, med gällande avgränsningar, att syftet med projektet har uppnåtts.

Figurer

1.1 Chalmers Vera Team och Verabilen [1] . . . 2
1.2 Motorn(mk6) som nu används i Vera . . . 3
1.3 Experimentmotor som använts under projektets gång - Honda GX35 . . . 3

2.1 Veramotorns uppbyggnad; cylinder(1), kolv(2), topplock(3), ventiler(4), kamaxel(5, dold), vevstake(6), vevaxel(7, dold) och vevhus(8), förbränningsrum(9) . . . 5
2.2 Fyrtaktscykelns fyra takter: insugs(1)-, kompressions(2)-, arbets(3)- och avgastakten(4) [2] . . . 6
2.3 Exempelgraf, vevaxelgrader-cylindertryck . . . 7
2.4 Gnista bildas mellan tändstiftets två elektroder[3] . . . 9
2.5 Illustration av tändspolens funktion . . . 9
2.6 Spridaren som används i motorn(16450-GEV-761). Nedre sidan ansluts mot motorn, längst upp anluts slang med trycksatt bränsle. I bilden syns även kontakten för strömanslutning. . . . 10
2.7 En vevaxelgivare som känner av kuggarna på hjulet. I nedre delen av bilden syns missing tooth. . . . 11
2.8 Exempel på en bränslemapp . . . 12
2.9 Enkel illustration av en H-brygga[4] . . . 13

3.1 Uppdelning av styrsystemet . . . 15
3.2 STM32F4 Discovery, utgör kärnan i ECU-modulen . . . 16
3.3 Halleffektsensor monterat vid pulshjul på testmotorns vevaxel . . . 18
3.4 Halleffektsensor monterad vid testmotors kamaxel . . . 18
3.5 .Kapseltyp TO-220[5] . . . 19
3.6 Kapseltyp TO-92[6] . . . 19
3.7 MAP-sensor(lufttryckssensor), BOSCH 0 261 230 031 . . . 19
3.8 Raspberry Pi[7] . . . 23
3.9 Glade användargränssnitt . . . 25
3.10 Testrigg för verifiering av styrsystemets funktion. Honda GX35(1), tändspole(2), STM32F4 Discovery(3), kamaxelsensor(4), spridare(5), bränsletank(6), drivelektronik för tändspole och spridare(7), MAP-sensor(8). . . . 27

4.1 Illustration av styrsystemet . . . 29
4.2 Simulinkmodell, toppnivå . . . 30
4.3 Kamrutin . . . 31
4.4 Vevrutin . . . 32
4.5 Rutin för bränsleberäkning . . . 34
4.6 Tändberäkningsrutin . . . 35
4.7 Konvertering till tidpunkter för tändsystemet . . . 36

4.8 Vevaxelposition i arbetscykeln 38
4.9 Rutin för positionsuppdatering 39
4.10 Räknerutin 40
4.11 Sensoravläsning 41
4.12 Periodisk hjälprutin 42
4.13 Loggrutin 43
4.14 Utsingalerna till H-bryggan vid simulation av Simulinkmodellen. 44
4.15 Utsignalerna till H-bryggan vid mätningar på verkliga systemet. 44
4.16 En spänningsregulator används för att försörja styrsystemet med rätt spänning 45
4.17 Signalanpassning för kam- och vev-axelsensor 46
4.18 Signalanpassning för temperatursensorer 47
4.19 Signalanpassning för MAP-sensor 48
4.20 Krets för att mäta batterispänning 49
4.21 Illustrerar hur knappar ansluts i styrsystemet 50
4.22 Drivkrets för tändning. IGBT 51
4.23 Kretsschema på styrsystemets h-brygga som möjliggör polvändning av strömmen till injektorn. 52
4.24 Ritning på kretskort som gjorts för styrsystemet 53
4.25 Uppkoppling på kopplingsdäck. STM32F4-Discovery(1), Spänningsregulator ECU(2), signalanpassning för kam- och vevaxelsensorer(3), drivkretsar för tändning och spridare(4), MAP-sensor(5), motortemperatursensorer(6), lufttemperatursensor(7). 54
4.26 HMI Arkitektur 55
4.27 Gränssnitt mot föraren 56
4.28 Webbsida för visning av HMI loggar 57
4.29 Webbsida för programmering av mappar 58
4.30 Karta på vilken Veras position visas 59
4.31 Graf för att visa hastighetsförändringar 59

Litteraturförteckning

[1] J. Eriksson, Development of android software for logging of enginge data for shell eco marathon. (2013).

[2] K. Nice, How automobile ignition works.
URL http://auto.howstuffworks.com/ignition-system2.htm

[3] R. Nave, Faraday's law and auto ignition.
URL http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/ignition.html

[4] B. Gaylord, Ignition coil function and testing ig-108 (2006-2012).
URL http://mgaguru.com/mgtech/ignition/ig108.htm

[5] S. Weiss, Electric fuel injector (efi) flow data table (2002-2014).
URL http://users.erols.com/srweiss/tableifc.htm

[6] E. Ramsden, Hall vs vr. which speed sensor is wrong for you? (2000).
URL http://www2.electronicproducts.com

[7] Full function engineering (2012).
URL http://fullfunctioneng.com/info/Hall%20vs%20VR.pdf

[8] Cantherm, How to choose and work with ntc thermistors.
URL http://www.cantherm.com/products/thermistors/choosing_ntc.html

[9] R. B. GmbH, Gasoline Engine Management, Vol. 3 edition, Wiley, 2006.

[10] C. Hacks, New 3g+gps shiels for raspberry pi tutorial.
URL http://www.cooking-hacks.com/documentation/tutorials/raspberry-pi-3g-gprs-gsm-gps

[11] Changpuak, Online heatsink calculator (2014).
URL http://www.changpuak.ch/electronics/calc_23.php

[12] W. Storr, Rc waveforms (2014).
URL http://www.electronics-tutorials.ws/rc/rc_3.html

A. Programmeringsmiljö för HMI

För att kunna fortsätta utveckla HMI-delen på ett bra och effektivt sätt krävs kunskap kring hur operativsystemet är uppstatt, vilka tredjepartsprogram som använts och hur man kompilerar och kör programmet.

Projektrelaterad miljö

Rotmappen för projektet är placerad i katalogen "/home/pi/VeraHMI". För C++ källkoden gjordes valet att ha .cc som filändelse. Bygga koden gör man genom att öppna terminalen i rotmappen och kör kommandot "make". Starta själva programmet gör man genom att köra kommandot "./bin". Om man vill kombinera dessa skriver man "make & ./bin", då kompilerar den först och sedan kör programmet. Make-filen är simpel men smart och bygger endast om källkodsfilerna som har ändrats, vilket är bra att vara medveten om. Om man använder något nytt paket eller bara behöver ändra flaggorna till kompileraren eller länkaren görs det här.

Filstruktur

I projektets rotmapp ligger följande filer och mappar:

- src - Roten för C++ källkod.
- www - Roten för webbgränssnittet för programmering av mappar och presentation av loggdata.
- layout.glade - Layouten för användargränssnittet.
- bin - Körbar binär för programmet.
- README.md - Readme fil för GIT.
- build - Temporär katalog för byggda filer.

Tredjepartsprogramvara

Nedan följer en lista med tredjepartprogram som används. Dessa har installerats via pakethanteraren apt-get. För konfiguration av dessa rekommenderas respektive programs hemsida för senaste informationen.

- sshd - SSH-server
- tightvnc - VNC-server
- Glade - https://glade.gnome.org

- lighttpd - Webbserver
- php5 - PHP
- mysql - Databas
- python - För programmering av ECU mappar
- isc-dhcp-server - DHCP server
- hostapd - Lokal access point

PHP och pythonscripten för programmering av mappar ligger i www/actions/ och är följande:

- save_mapping.php
- write.py
- load_mapping.php
- read.py

B. Kravspecifikation

Detta är en kravspecfikation av styrsystemets funktionallitet sammanställd av Chalmers Vera Team.

Sensorer

1. Styrsystemet skall ha stöd för följande sensorer:
    a. Varvtalsgivare CAD
    b. Lägesgivare kamaxel
    c. Lägesgivare svänghjul
    d. Temperatursensorer
        i. Cylinder
        ii. Topplock
        iii. Vevhus
    e. Luftsensorer
        i. Tryck
        ii. Temperatur
        iii. Fukthalt
    f. Knacksensor
    g. Spänningsgivare för batteriet

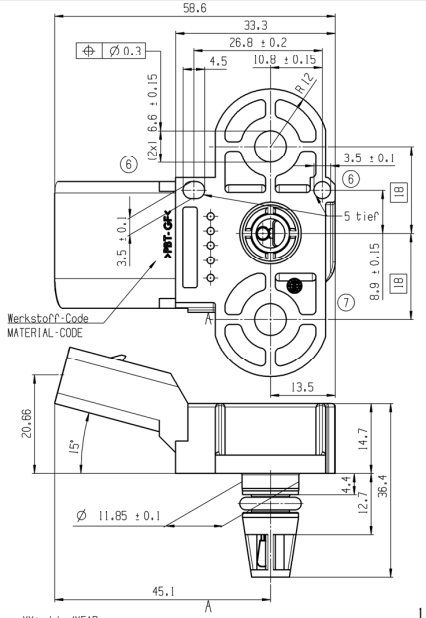
Motorstyrning

1. Styrning av tändning ska ske på följande sätt:
    - CAD mot varvtal
    - Tändfönster mot varvtal
2. Styrning av bränsle ska ske på följande sätt
    - CAD mot varvtal
    - Massa mot varvtal
3. Styrsystemet ska ha startmappar för tändning och bränsle som är aktiva i ställbart antal varv efter motorstart.
4. Styrsystemet ska kunna polvända strömmen till solenoiden i injektorn för att få mer exakt kontroll av injektorn och därmed förbättrad kontroll av bränsleflödet.
5. Styrsystemet ska ha ställbart startvarvtal
6. Mapparna för tändning och bränsle ska vara baserade på varvtal och ha en upplösning på 20 rutor
7. Styrsystemet ska ge möjlighet att ställa vid vilket varvtal bilen ska stänga av.
8. Styrsystemet ska ha maxvärden för motortemperatur och varvtal där motorn ska stängas av som skydd
9. När motorn stängs av ska bränsle brytas, inte tändningen.
10. Motorn ska aldrig stängas av mitt i en insprutningscykel
11. Styrsystemet ska analysera motorns acceleration och meddela att den följer, eller inte följer, en förväntad acceleration.
12. Styrsystemet skall kompensera för sjunkande batterispänning

13. Styrsystemet bör vara förberett för motorer med solenoidstyrda ventiler

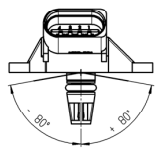
Övrigt

1. Styrsystemet ska larma och ge felkoder vid fel
2. Man ska kunna spara motordata under ett race
3. Förare ska lätt kunna se varvtal, varvtid och felkoder
4. Skall finnas mjukvara där mappar och motorparametrar kan ställas in och alla värden kan läsas enkelt
5. Styrsystemet skall förpackas i en vattentät låda med automotivekompatibla kontaktstycken
6. Styrsystemet bör vara kompakt och ha låg vikt
7. Motordata bör kunna avläsas via internet under körning



Einbaulage im Fahrzeug:
Stutzen nach unten gerichtet:
Empfohlene Lage: 0...± 60° in allen Richtungen zur Senkrechten
Einbaulage ± 80° zulaessig

MOUNTING POSITION IN THE VEHICLE:
ORIFICE DIRECTED DOWNWARDS
RECOMMENDED POSITION: 0...± 60°
FROM VERTICAL
MOUNTING POSITION ± 80° PERMISSIBLE



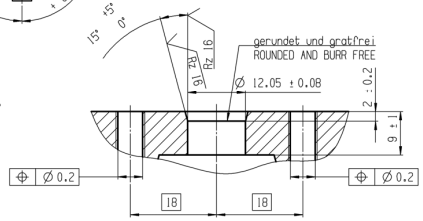
4-poliger VW Stecker, VW 1JO 900 962, schwarz, Codierung 1.
Steckkontakte nach VW N 905 854.04, Kontaktflaechen vergoldet.
Stecksystem in Verantwortung des Kunden.

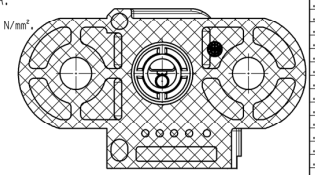
4 PIN VW CONNECTOR, VW 1JO 900 962, BLACK, CODE 1.
TERMINALS ACCORDING TO VW N 905 854.04, CONTACT SURFACE GOLD PLATED.
CONNECTOR IN RESPONSIBILITY OF THE CUSTOMER.

Anbaustelle
HOLE FOR PRESSURE SENSOR FITTING

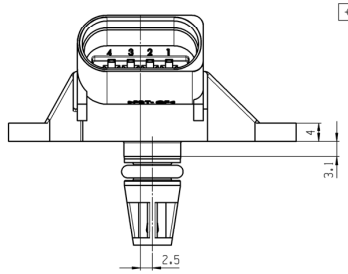
Maximaler Durchmesser der Befestigungsschraube innerhalb des Befestigungslochs: 6 mm.
Unterlegscheibe ø 10 mm oder entsprechenden Schraubenkopfdurchmesser verwenden.
Selbstsichernde Schraube verwenden.
Maximal zulaessige Druckbelastung des Befestigungsflansches: 20 N/mm².

MAXIMAL DIAMETER OF THE SCREEN IN THE INTERIOR OF THE MOUNTING HOLE: 6 mm.
USE WASHER DIA. 10 mm OR ACCORDINGLY SCREW HEAD DIAMETER.
USE SECURITY SCREWS.
MAXIMAL PERMISSIBLE PRESSURE ON THE MOUNTING FLANGE: 20 N/mm².

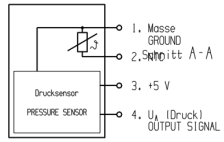




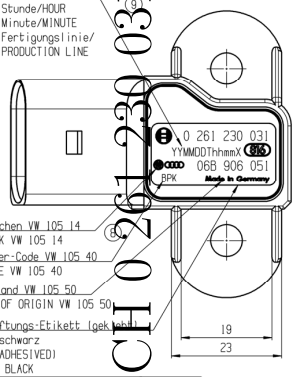
Auflageflaeche fuer Sensor.
Sollte (von Rippen) unterstuetzt sein.
AREA TO SUPPORT SENSOR.
SHOULD BE SUPPORTED (BY RIBS).



Steckerbelegung
CONNECTOR DETAILS



YY: Jahr/YEAR
MM: Monat/MONTH
DD: Tag/DAY
T: Trennzeichen/ SEPARATOR
hh: Stunde/HOUR
mm: Minute/MINUTE
X: Fertigungslinie/ PRODUCTION LINE



Warenzeichen VW 105 14
TRADEMARK VW 105 14
Hersteller-Code VW 105 40
MFR.-CODE VW 105 40
Herstellland VW 105 50
COUNTRY OF ORIGIN VW 105 50
Beschriftungs-Etikett (gek.lebl.)
Farbe: schwarz
LABEL (ADHESIVED)
COLOUR: BLACK

O-Ring ø 7.59 ± 0.16 x ø 2.62 ± 0.09
Werkstoff: HNBR Shore A
Farbe: Gruen
Zugehoeriger Nutgrunddurchmesser: 7.9 ± 0.15

>400 ≤ 1000	.	.
>120 ≤ 400	.	.
> 30 ≤ 120	± 1.0	.
> 6 ≤ 30	± 1.0	.
≤ 6	± 1.0	.
Winkel Angle	± 5'	.
	fertig finished	roh raw

Allgemeintoleranzen fuer Nennmasze ohne Toleranz.
General tolerances for nominal dimensions without specified tolerance.

Fa. Co. .
Fa. Co. .
Fa. Bosch Co. Entnach 6 .
zugelassener Entwickler/Hersteller Approved designer/supplier
Bausterfreigabe erteilt/Datum Type release granted/date

Bausmustergenehmigung durch TE erforderlich
Type approval by TE required

Bemerkungen/ Notes

Oberflaechenrauheit
Surface Finish
VW 137 05

Werkstuckkanten
Edge Finish
VW 010 88

Alle Masze gelten fuer das Fertigteil einschliesslich metallischen Oberflaechenschutzes.
All dimensions apply to the finished product including material surface protection.

Rev.-Index/ Angabe/Aenderungen
FURTHER/ FOREIGN LANGUAGE OFFER DWGS

1. Einbauhinweis

Der Sensor ist fuer den Anbau an eine ebene Flaeche am Saugrohr von Kraftfahrzeugen ausgelegt.
Der Druckstutzen und der Temperatursensor ragen gemeinsam ins Saugrohr und werden durch einen O-Ring zur Atmosphaere abgedichtet. Durch einen geeigneten Einbau im Fahrzeug (Druckentnahmestelle oben am Saugrohr, Druckstutzen nach unten geneigt, usw.) ist sicherzustellen, dass sich kein Kondensat in der Druckzelle anlagert.
Die Bohrung im Saugrohr und die Befestigung sind nach Angebotszeichnung auszufuehren, so dass ein dauerhaft dichter Sitz am Stutzen sowie die Bestaendigkeit gegen das Messmedium gewaehrleistet sind.
Steckerkonzeption nach Kundenvorgabe.
Der Kunde stellt die Dauerfestigkeit in eigener Verantwortung sicher.
Fuer Maengel am Stecksystem keine BOSCH-Gewaehrleistung.

2. Daten

2.1 Grenzdaten Drucksensor

Groesse	Zeichen	Wert	Einheit
Speisespannung	$U_{S,max}$	16	V
Druck	$p_{abs,max}$	500	kPa
Lagertemperatur	t	-40/+130	°C

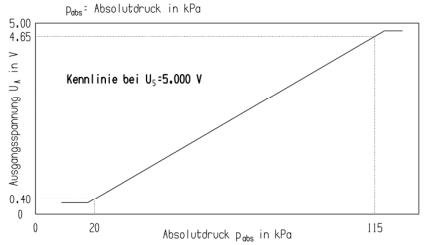
Der Ausgang ist kurzschussfest gegen 0 V bzw. 5 V. Der Sensor ist verpolsicher fuer 5 min bei Raumtemperatur, sofern der Maximalstrom auf 0.3 A begrenzt ist.

2.2 Kenndaten Drucksensor

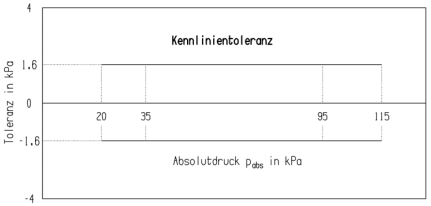
Groesse	Zeichen	Wert min	Wert typ	Wert max	Einheit
Druckmessbereich	p_{abs}	20		115	kPa
Betriebstemperatur	t	-40		130	°C
Speisespannung	U_S	4.5	5.0	5.5	V
Stromaufnahme bei U_S: 5 V	I_S	6.0	9.0	12.5	mA
Laststrom am Ausgang	I_L	-1.0		0.1	mA
Lastwiderstand nach U_S oder nach Masse	$R_{pull-up}$ / $R_{pull-down}$	5 / 50	680 / 100		kOhm
Ansprechzeit	$T_{10/90}$			1.0	ms
untere Begrenzung bei U_S: 5 V	$U_{A,min}$	0.25	0.3	0.35	V
obere Begrenzung bei U_S: 5 V	$U_{A,max}$	4.75	4.8	4.85	V
Ausgangswiderstand nach Masse, U_S offen	R_{lo}	2.4	4.7	8.2	kOhm
Ausgangswiderstand nach U_S, Masse offen	R_{hi}	3.4	5.3	8.2	kOhm

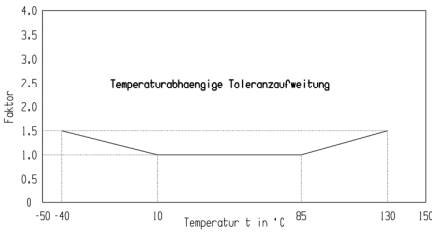
2.3 Uebertragungsfunktion Drucksensor: $U = (c_1 * p_{abs} + c_0) * U_S$

mit U_A : Signalausgangsspannung in V
U_S : Speisespannung in V
p_{abs}: Absolutdruck in kPa
c_0 : -9.4/95
c_1 : 0.85/95 kPa^{-1}



2.4 Genauigkeit Drucksensor





Dokumentenart/ DOCUMENT TYPE AGZ	Nicht tol. Masse NON TOLERANCED DIMENSIONS ± 1 mm, ± 5 °	ISO E	Massstab SCALE 2:1 (1:1)	Gewicht WEIGHT ca. 25g
			Angebots-zeichnung	OFFER DRAWING

Gr.St.	Datum/ DATE	Name/ NAME
VOLUME	ges/DRA 20.04.1998	Je
	ges/CHK 24.04.1998	Ho
	ges/APP 24.04.1998	Ku

Original: 213-51/ENS

Benennung/ TITLE: Druck-/Temperatursensor
PRESSURE-/TEMP.SENSOR

Nr./ NO. A 261 260 285-003 de/en Blatt/ SHEET 2 BLATT/SH

Ers.f./ REPLACES A 261 260 285-003 17.04.1999 Ers.d./ REPLACED BY .

Nr. NO	Aenderung CHANGE	ges. DRN.	gueltig VALID	gepr. CHK.
9	FD gegendert	Hk	03.12.2008	Al
8	Werkskennzahl geaend.	Al	13.05.2005	Hh
7	Ueberarbeitet	Fe	07.02.2002	Ho
6	A2BEH10580	Ho	16.06.1999	Je

DIN A1 LG

B 261 260 285	0 261 230 032 (EA) / 0 261 230 031 (KH)	0 261 230 031 / 0 261 230 031	06B 906 051	Y 261 K26 055-003	20...115 kPa
Entwurfs-Nummer DRAFT NUMBER	Bestell-Nummer ORDER NUMBER	Typ-Aufschrift TYPE DESIGNATION	Kunden-Teile-Nummer CUSTOMER PART NUMBER	Techn. Kundenunterlage TECHNICAL SPECIFICATION	Druckbereich PRESSURE RANGE

Form FE 504 - 1196 Form TE 04989
Bezugsmasze / Ref. dim. | Kontrollmasze / Control dim. | Vorrichtungsmasze / Fixture dim. | Ausfallmasze / Temporary dim. | Pruefmasze / Test dim. | Pruefmasze mit Bewertung / Dim. to be inspected | Theoretische Masze / Basic dim. | Werkstoff-Code / Material code

D/ESM 151.2001 A A1

Nr. No.	Feld Section	Datum Date	Geaendert Changed	Genehmigt Appr.	Beschreibung der Aenderung und Aenderungsterminschluessel / Revision Record and Change date code
9		20.04.1998	Je		FD gegendert
8		13.05.2005	Hh		Werkskennzahl geaend.
7		07.02.2002	Ho		Ueberarbeitet
6		16.06.1999	Je		A2BEH10580

Unterlagen References	
VW 105 14	06.89
VW 105 40	06.92
VW 105 50	04.91
VW 105 60	12.91
VW 801 01	01.98
VDA 260	.

A 261 260 285-003

Zeichnungs-Ausfuehrung entspricht nicht in allen Punkten den VW-Zeichnungsnormen.
Drawing-details do not meet VW-standards for designing in all cases.
NUR FUER FIRMA UND INTERNEN GEBRAUCH / FOR SUPPLIER AND INTERNAL USE ONLY
Robert Bosch GmbH
Genehmigung von Erstlieferung und Aenderung nach VW 01155. Approval for initial supply and modifications according to VW 01155. The English translation is believed to be accurate. In case of discrepancies the German version shall govern.

Typ-Pruef-Bk. Type-Appr.-Doc. / CAD-System und Verwaltungssystem-Schluessel/ CAD-System and Administration code

Werkstoff/ Material: PDT-GF
Konst.-Verantwortg. Design Resp.
Name
Werkstoffbehandlung/ Material treatment
Halbzeug/ Semi-finished product
Ent.-Nr. Layout No.
Abt. Dept.
Oberflaechenschutz / Surface protection: OFI-
Tel.
Gewicht/ Weight (g): gerechnet/ calculated | gewogen/actual: Versuch prototype | Fertigung production
EA-Nr. Eng.Proj.No.
Gez./Drawn: Fa. Bosch 20.04.98

Sicherh.Dok. Safety Doc.
Benennung: Druck-/Temperatursensor (DS-S2-TF)
Title: PRESSURE-/TEMPERATURE SENSOR (DS-S2-TF)
Maszstab/ Scale: 2:1 (1:1)
Teil-Nr. Part-No.: 06B 906 051
Format A1 0.Size
Blatt/ Sheet 1 von/ of 2

© Alle Rechte bei Robert Bosch GmbH, auch fuer den Fall von Schutzrechtsanmeldungen. Jede Verfuegungsbefugnis wie Kopier- und Weitergaberecht, bei uns.
© THIS DRAWING IS THE EXCLUSIVE PROPERTY OF ROBERT BOSCH GMBH. IT MUST NOT BE COPIED (UNLESS AUTHORIZED) OR GIVEN TO A THIRD PARTY NOR MISUSED IN ANY OTHER WAY.