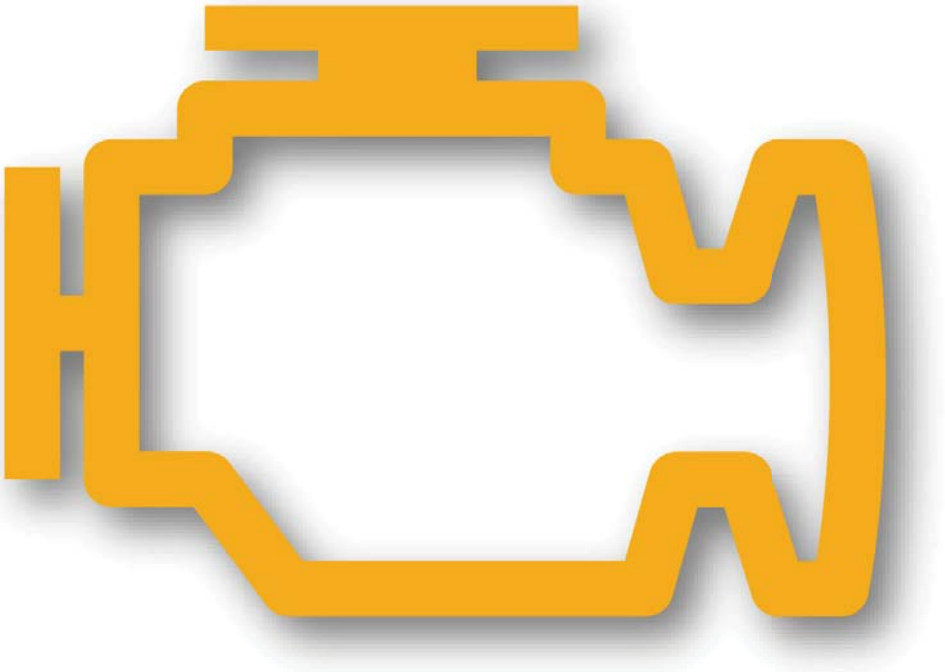
CHALMERS



Utredning och automatisering av verifieringsprocesser

Motorstyrsystemets mjukvara på Volvo Cars

Examensarbete inom högskoleingenjörsprogrammet Elektroingenjör

Tommie Heikkinen
John Tagesson Ritz

Handledare: Manne Stenberg

Examinator: Bertil Thomas

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

Figur framsidan: MIL-lampa, Wikimedia commons, 2009, http://commons.wikimedia.org/wiki/File:Motorkontrollleuchte.svg (2015-06-01)

Utredning och automatisering av verifieringsprocesser

Motorstyrsystemets mjukvara på Volvo Cars

Tommie Heikkinen
John Tagesson Ritz

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

Förord

Detta examensarbete är den avslutande delen inom ramen för utbildningen på Elektroingenjörsprogrammet, 180 hp, vid Chalmers Tekniska Högskola. Arbetet omfattar 15 hp inom Institutionen för Signaler och System.

Under vårterminen 2015 har vi, på uppdrag av Volvo Cars, befunnit oss på Volvo Cars avdelning Research and Development (R&D) där vi haft förmånen att arbeta med intressanta system och kunnigt folk inom området. Det har varit en mycket givande och lärorik tid, samtidigt som det har gett oss en värdefull inblick i fordonsindustrin.

Vi vill rikta ett särskilt tack till Gustaf Agrenius som på ett mycket engagerat sätt varit vår handledare och som har bistått oss med information såväl som tagit sig tid att svara på våra frågor och kommit med värdefulla synpunkter och uppskattade kommentarer. Ett stort tack riktas också till Erik Barrefors som bidragit med sin expertis inom området för konstruktion och kodning på ett pedagogiskt sätt. Vi vill också framföra ett varmt tack till de personer på Volvo Cars som under arbetets gång bidragit med såväl kunskap som stöd, Daniel Tidholm, Per Gustafsson (PG), Mattias Eriksson och Topi Heikkinen. Vi vill även tacka Martina Steineck för att vi fick möjlighet att genomföra detta examens arbete på Volvo Cars.

Ett stort tack riktas också till vår handledare Manne Stenberg, vid Institutionen för Signaler och System, för feedback och vägledning genom detta arbete samt för visat sitt intresse genom att besöka oss på Volvo Cars.

Göteborg, juni 2015.

Tommie Heikkinen
John Tagesson Ritz

Sammanfattning

På Volvo Cars utförs idag verifieringen av diagnosmjukvaran i motorstyrenheten i stort sett helt manuellt. För att minska det manuella arbetet och därmed öka effektiviteten vill Volvo få möjligheten att flytta verifieringen från en verklig bil, så som det görs idag, till en Hardware-in-the-loop (HIL) simulator. För att styra HIL-systemet och andra programvaror som behövs för verifiering används programspråket Python. De processer som Volvo vill få automatiserat är verifieringsprocesser av diagnoskärnan i motorstyrsystemet. För att ge Volvo en överblick över vad som går att automatisera med hjälp av HIL-systemet och vilka begränsningar som finns, genomfördes först en utredning. För att arbetet inte skulle bli för omfattande infördes en del avgränsningar, endast två processer utreddes och därefter automatiserades en av dessa processer. Resultatet från utredningen pekar på att automation av både felkodsverifiering för elfel och J1699-3 verifiering är möjlig. Processen som valdes att automatiseras kräver inga större insatser vad det gäller ny hårdvara eller nya funktioner i Python. Däremot för den andra utredda processen behövs lite mer arbete då kraven för denna verifiering är striktare och verifieringen blir därmed mycket mer omfattande, men är fullt möjlig att genomföra. Programmet som konstruerades utformades så generiskt som möjligt för att möjliggöra vidareutveckling av programmet och utökning till fler områden. För att konstruera programmet skapades flera nya funktioner som till exempel funktioner som kan skicka styr-signaler till testobjektet för att göra det möjligt att styra den simulerade bilen. Det konstruerade programmet kan genomföra en enkel felkodsverifiering för elfel men det finns fortfarande flera delar i programmet som behöver vidareutvecklas.

Abstract

Today verification of diagnostic control modules are performed manually at Volvo Cars. To reduce the amount of manual work and therefore improve the efficiency, Volvo wishes to move the verification process from a real car to the Hardware-in-the-loop simulator (HIL). To be able to control the HIL-system and other software that is needed in the verification process the program language Python is used. The verification processes that Volvo wants to automate are verification-processes of the diagnostic core in the engine control system. To give Volvo an overview of the HIL-system and its capabilities for automation of the processes, an investigation is made. Some limitations are introduced, where only two of the processes are investigated and then one of them is chosen to be automated. The results from the investigation shows that it is possible to automate the verification processes. The process that was chosen to be automated does not require any new hardware to be introduced. On the other hand the other processes needs a little more work in order to be automated, since the requirements for this process is much stricter and therefore much more complicated to design. The program was designed as generic as possible in order to allow further development to be as easy as possible. The construction of the program required many new functions to be written such as functions that are able to transmit control signals to the test object in order to allow the program to control the vehicle. The program that was constructed is able to perform a simple fault code verification for electrical faults although further developments can be made.

Innehållsförteckning

Beteckningar

CARB – **C**alifornia **A**ir **R**esources **B**oard

DID – **D**ata **Id**entifier

DSA – **D**iagnostic and **S**oftware Download **A**pplication

DTC – **D**iagnostic **T**rouble **C**ode

ECM – **E**ngine **C**ontrol **M**odule

ECU – **E**lectronic **C**ontrol **U**nit

ETAS – **E**ngineering **T**ools, **A**pplication and **S**ervices

ETK – Emulation Probe (**E**mulation **T**ast**k**opf)

FIU – **F**ault **I**njection **U**nit

ISO – **I**nternational **O**rganization for **S**tandardization

MID – **M**onitor **Id**entification

MIL – **M**alfunction **I**ndicator **L**amp

OBD – **O**n-**B**oard-**D**iagnostics

PID – **P**arameter **Id**entifier

SAE – **S**ociety of **A**utomotive **E**ngineers

TID – **T**est **Id**entifier

UDS – **U**nified **D**iagnostic **S**ervices

VED – **V**olvo **E**nvironmental **D**iesel

VEP – **V**olvo **E**nvironmental **P**etrol

WUC – **W**arm **U**p **C**ycle

1 Inledning

På Volvo Cars finns behov av att utöka användningen av Hardware-in-the-loop (HIL) till simulering av bilar för testning av diagnosmjukvaran i motorns kontrollenhet, Engine Control Module (ECM). Idag utförs testning av motorns kontrollenhets mjukvara till stor del manuellt vilket är tids- och resurskrävande. Därmed finns det behov av att automatisera dessa arbeten. De processer som Volvo önskar att automatisera är; verifiering och mätning i ECM-enhetens kärna, felkodsverifiering och J1699-3 provning.

Volvo vill utreda om HIL-systemet går att använda till dessa processer och ta fram eventuella brister eller utvecklingsområden. Efter utredningen ska ett program skapas som ska utföra det idag manuella arbetet i det närmaste automatiskt.

Utredningen börjar med informationssökning och studier av program- och hårdvara som kommer att användas, som exempelvis HIL-systemet, parallellt med inlärning av de olika manuella processerna. Kontroll av vilka krav de olika manuella processerna ställer på HIL-systemet utförs och därefter utreds vad som går att automatisera med hjälp av HIL-systemet och vilka funktioner som i dagsläget finns färdiga. Efter utredningen väljs en process som ska automatiseras och vilken programvara som är lämplig för arbetet. Därefter konstrueras ett program som utför arbetet där stor vikt läggs på programmets struktur för att slutprodukten ska bli så generisk och lättförståelig som möjligt. Sist analyseras och utvärderas resultatet och brister i systemet åskådliggörs för att belysa vad som krävs för att kunna vidareutveckla programmet och bredda användningen till andra områden.

1.1 Bakgrund

På Volvo Cars utförs idag verifiering av diagnosmjukvaran manuellt. Detta arbete är tidskrävande och Volvo har behov av att automatisera detta genom simuleringar av bilar i hårdvara med hjälp av HIL-systemet. Det finns idag en del färdigutvecklade funktioner som kan utföra vissa delar av automatiseringen men Volvo har behov av att sammanföra och utöka dessa funktioner för att kunna utföra mer omfattande och utförligare verifieringar av dagens manuella arbete. Således kan en utveckling av HIL-systemets funktionalitet medföra ökad effektivitet i mjukvaruutvecklingen och därmed ökad produktivitet i verksamheten.

1.2 Syfte

Syftet med arbetet är att automatisera en verifieringsprocess på Volvo Cars och därmed utöka Volvos användande av HIL-systemet. Tre processer har tagits fram för utredning av vad som är möjligt att automatisera varav en ska väljas och ett prototypprogram som utför arbetet med minimal interaktion från användaren ska konstrueras.

1.3 Avgränsningar

Utredning av alla tre processer skulle troligtvis leda till ett tämligen omfattande arbete vilket skulle resultera i ett alltför enkelt program istället för ett noggrant och generiskt utformat program. Därför har avgränsningar gjorts och därmed har inte verifiering av kärnan under-sökts. Alltså har endast utredning av felkodsverifiering och J1699-3 provning gjorts. Upp-giften var sedan att efter utredningen välja en process och konstruera ett användarvänligt program.

Fokus ska ligga på att åstadkomma ett specifikt men generiskt program där implementering av endast en motortyp realiseras. Detta med avsikten att vidareutveckling för ytterligare motortyper ska vara enkelt att genomföra.

1.4 Precisering av arbetsuppgiften

Nedan är några punkter uppställda med frågeställningar som kommer att bli aktuella under projektets gång:

- Hur ser processen ut i dagsläget?
- Vad krävs för att genomföra ett test?
- Finns det som behövs tillgängligt?
- Vilken programvara kan användas för att automatisera processen?
- Vilka krav finns på slutprodukten?
- Hur verifieras säkerheten i testet så att det utförts korrekt?
- Hur ska resultatet kontrolleras och rapporteras?

2 Metod

Den valda metoden är generell till specifik, det vill säga att först utreddes de olika områdena genom en undersökning av vad som var möjligt och vad som önskades automatiseras. Där-efter valdes en specifik del och en plan togs fram för hur ett prototypsystem skulle kunna konstrueras och därefter vidareutvecklas till att ersätta dagens manuella arbete. Utredningen av hur arbetet skulle utföras kartlades genom möten med experterna på Volvo där diskussion kring de efterfrågade processerna undersöktes (från beskrivning av Volvo). Därefter utreddes hur dagens manuella arbete utförs och studier av program- och hårdvara gjordes. Detta för att det sedan skulle vara möjligt att ta fram idéer för utformning av automatiseringen. För att er-hålla kunskap för den tekniska bakgrunden gjordes informationshämtning från Chalmers bibliotek. Vid konstruktion av programmet användes sekventiell kodning. Detta då det gör att det är lättare att verifiera att varje steg har gjorts innan nästa steg påbörjas.

3 Teknisk bakgrund

För att få en bild av vilken utrustning som ska användas i de verifieringsprocesser Volvo önskar automatisera krävs fördjupning i ett flertal olika områden. Då processerna som ska utredas för närvarande utförs manuellt finns behov av att konstruera ett program som kan utföra detta arbete. Det är nödvändigt att förstå viss hårdvara såväl som programvara för att möjliggöra konstruktion av programmet som efterfrågas.

3.1 Hårdvara

Den aktuella hårdvaran som ska studeras är Hardware-in-the-loop och Electronic Control Unit modulen. Denna hårdvara är vad som kommer att styras med det program som ska konstrueras för att automatisera processen.

3.1.1 Hardware-in-the-loop

Hardware-in-the-loop, förkortat HIL, används för att simulera och utveckla integrerade system. En modell av systemet skapas i HIL-simulatorn och eventuella svår-simulerade hårdvaror kopplas in fysiskt. Att vissa hårdvaror kopplas in fysiskt beror på att de, om inte simulering är möjlig, kan orsaka oönskade fel i systemet. Efter att modellen är skapad kan mätningar och tester utföras på testobjektet.

Figur 1, Full-size HIL Photo: dSpace [5]

Att testa ett system i en HIL-simulator underlättar konstruktionen av systemet då tester kan göras innan ett komplett fysiskt system existerar. På detta sätt kan tid och resurser sparas. I ett stort komplext system där ett fel i designen kan leda till personliga och materiella skador kan en HIL bidra till att ett i princip färdigt system kan finnas att tillgå redan första gången det testas i ett färdig konstruerat system. Dessutom kostar inte ett fel i designen av mjukvaran lika mycket i de första stadierna av konstruktionen som om felen upptäcks när en färdig produkt är producerad.

I de flesta system idag används någon form av mjukvara för att styra och kontrollera systemet. När ett system ska testas används ofta en HIL-simulator för att simulera maskinen eller systemet och på det sättet ”lura” mjukvaran att tro att den styr ett fysiskt system. Eftersom samma kabelbunt används för HIL-simulatorn som för den fysiska maskinen kan exempelvis en kontrollenhet flyttas utan problem mellan simulatorn och maskinen eller systemet. [14][21]

3.1.2 Electronic Control Unit (ECU)

I en modern bil finns ett antal olika elektroniska kontrollenheter så kallade ECU-enheter. Varje ECU-enhet har ett specifikt område i bilen att kontrollera som till exempel motorn eller transmissionen. ECU-enheten fungerar som ett reglersystem med åter-koppling. Den samlar in mätdata från olika enheter och erhåller styrdata från en annan ECU eller från föraren. Informationen processas och enheten skickar sedan ut styr-signaler till exempelvis ett spjäll eller tändningssystemet. De givare eller signaler som mäts innehåller en mängd olika para-metrar däribland motorvarvtal, bilens hastig-het, laster, temperaturer och flöden. En givare kan användas för många olika uppgifter som exempelvis att ge föraren information om bilens status. De kan även användas i diagnostiskt syfte genom att spara undan data från en mängd olika givare vid ett feltillfälle, så kallade *freeze frames*, för att verkstäder ska kunna följa upp vad som hänt vid feltillfället och utreda vad som kan ha orsakat felet.

Figur 2, ECM-enhet, uppe till höger kan ETK anslutningen ses och längst ner finns anslutningar för signalkablage

En av alla de ECU-enheter som finns i en bil är Engine Control Module (ECM). ECM-enheten har två uppgifter, den arbetar som en kontrollenhet och som ett diagnosverktyg. Den fungerar som ett styrsystem för motorn och ser till att motorn levererar den begärda effekten samt kontrollerar att alla sensorer fungerar som de ska. Samtidigt utför den kontinuerligt en mängd olika diagnoser. Detta för att säkerställa att motorn går så effektivt som möjligt för att utsläppen ska bli så låga som möjligt.

Den del som arbetar som en kontrollenhet mäter enkelt förklarat luftflödet till motorn samt hur bra motorn tänder och beräknar därefter rätt tändtidpunkt och bränslemängd (så att rätt luftflöde uppnås) för att motorn ska producera rätt effekt. Vid beräkningen av rätt bränslemängd och tändtidpunkt använder sig ECM-enheten av så kallade *maps* eller *look-up tables*. Det är tabeller med värden för tändtidpunkt samt bränslemängd vid olika motorvarvtal och tryck i insugsröret. Ofta använder man sig av flera olika uppsättningar av tabeller som används vid olika förhållanden som till exempel vid kallstart eller körning på hög höjd. Det går också att, istället för att använda sig av tabeller, använda sig av komplexa modeller av motorn och på det sättet beräkna de rätta värdena vid olika arbetspunkter för motorn.

Den diagnostiska delen av ECM-enheten arbetar med att kontrollera olika mätningar av signaler och på så sätt övervaka systemet. När ECM-enheten upptäcker ett bestående fel lagrar den felkoden i felkodsminnet som därefter kan utläsas med hjälp av ett så kallat Scantool-verktyg. Ett Scantool-verktyg är ett instrument som kan kommunicera över CAN-bussen med ECU-enheterna i bilen via så kallade Service-funktioner. Det finns en mängd olika diagnosmonitorer som kontinuerligt utför diagnoser och som övervakar specifika signaler för att kontrollera om det finns något fel. Hittas ett fel lagras de som felkoder i minnet.

Felkoder som är emissionsrelaterade är lagkravsstyrda och kan läsas ut med ett så kallat Scantool-verktyg. Några viktiga monitorer som övervakar emissionsrelaterade delsystem, och därmed är lagkravsstyrda, har egna så kallade Monitor Identifiers (MID). Under dessa finns ett flertal diagnosmonitorer med identifierade namn, kallade Test Identifiers (TID). Exempelvis skulle en MID kunna vara en diagnos för katalysatorn, där det finns ett flertal TID som övervakar katalysatorn i realtid. Alla MIDars testresultat (Service $06) kan läsas ut och ger på det sättet information om hur nära monitorn har varit att sätta en felkod.

Utöver dessa större diagnosmonitorer finns det hundratals andra diagnosmonitorer som inte har en känd MID och därmed kan man inte läsa ut deras testresultatet med Scantool. I dessa fall finns endast felkod som identifierar felet och kan berätta om felet finns eller inte finns.

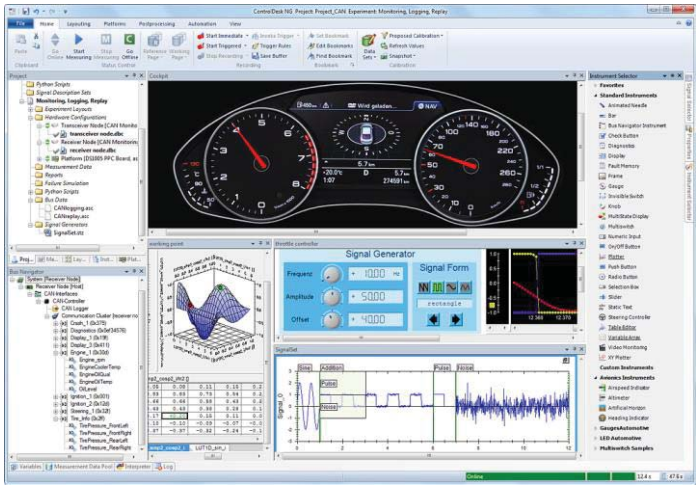
För att Volvoverkstäder skall kunna mäta olika parametrar från en ECU används så kallade Data Identifiers (DID). Dessa DIDar innehåller data om allt i en bil. Det kan exempelvis vara information om hur lång tid bilen har kört över ett visst varvtal eller vad en givare erhållit för mätvärde. På Volvo används Volvos programvara Diagnostic and Software Download Application (DSA) för att läsa ut dessa.[6]

3.2 Programvaror

De programvarorna som undersöks är ControlDesk, INCA och Python. Dessa program används vid automatiseringen av verifieringsprocesserna.

3.2.1 ControlDesk

För att kunna injicera elektriska fel i ECU-enheten behövs ett program från dSpace som heter ControlDesk. Detta program kan styra HIL-systemet som i sin tur kan bryta kretsar (simulering av öppenkrets), koppla mot pluspolen eller minuspolen/jord.



Figur 3, Screenshot från ControlDesk. Photo: dSpace [4]

Den del i HIL-systemet som simulerar fel genom att bryta kretsar är FIU-enheten och är uppbyggd av ett antal reläer. Det är bland annat dessa reläer som styrs med hjälp av ControlDesk. Därutöver finns det möjlighet att skapa modeller som kan simulera signaler och skicka dem till exempelvis ECM-enheten så att den tolkar det som om det vore bilen. Dessa simulerade signaler kan till exempel vara bromspedalen, gaspedalen, motorhastigheten, motorvarvtalet och mycket mer.

3.2.2 INCA

ETAS är ett företag som utvecklar olika typer av inbyggda system för industrin och då i synnerhet fordonsindustrin.[1] Mätprogrammet INCA används för att bland annat ladda ner mjukvara till ECM-enheten. Varje bilmodell utvecklas för olika marknader vilket medför olika krav för hur ECM-enheten ska uppföra sig. Med hjälp av INCA kan specifik mjukvara laddas till ECU-enheten. Detta program används mycket på Volvo Cars och är ett program som kan, utöver nedladdning av mjukvara till ECM-enheten, läsa av värden från ECU-enheten med hjälp av ett ETAS mätverktyg (ES591.1). För att kunna erhålla data från ECU-enheten krävs det att en speciell enhet med en inbyggd *Emulation Probe* (ETK) används. ETK-modulen möjliggör kommunikation med ES591.1-modulen som via en Ethernet-anslutning är ansluten till datorn.

INCA möjliggör även kalibrering av styrenheten. Kalibrering innebär att man anpassar styrparametrar i programmet så att önskade funktioner uppnås. Men kan användas för att manipulera på olika sätt för att kunna testa olika funktioner i bilen. Dessa test kan exempelvis vara felkodsverifiering där specifika felkoder ska sättas vid specifika förhållanden och då kan kalibrering krävas för att uppnå önskade förhållanden. Därefter kan användaren exempelvis läsa ut vilka felkoder som sitter genom att läsa ur DTC-minnet och på så sätt verifiera att ECU enheten arbetar som den ska.[7]

3.2.3 Python

Python är ett högnivå-programspråk som används för både objektorienterad och funktionell programmering. Det är ett enkelt språk i jämförelse med exempelvis C då mycket av dess struktur är lättare att handskas med. I Python används varken semikolon för att avsluta rader eller klammerparentes för att definiera hur stor funktionen är, istället används tabbar för att visa vilken funktion raderna tillhör.[15]

3.3 Begränsningar

Under konstruktionen av programmet kan en del begränsningar uppkomma till följd av att så många olika programvaror ska kommunicera med olika enheter. Exempelvis kan hastigheten på hårdvaran i förhållande till datorn vara låg vilket medför att programmet begränsas och fördröjningsfunktioner i programvaran behöver implementeras. Därutöver finns det fördröjningar i kärnan som säkerställer att ett fel är bestående eller att en signaländring inte är en störning, så kallade debounce-tider. Dessa fördröjningar kan påverka hastigheten i programmet då ett injicerat fel och styrsignaler måste sitta en viss tid för att de ska upptäckas och inte uppfattas som glitchar.

3.4 Nätverk i fordon

I dagens fordon används många olika sorters nätverk för att skicka information mellan kontroll-enheterna. För att få en uppskattning av vad som krävs i arbetet är det nödvändigt att närmare studera några av de nätverk som används i bilar på Volvo Cars.

3.4.1 Local Interconnect Network (LIN)

Local Interconnect Network, eller LIN, är ett protokoll för busskommunikation som huvud-sakligen är framtaget för fordonsindustrin. Bussen som används är ett 1-trådsnätverk med en masternod och upp till 15 slavnoder. En fördel med LIN-bussen är att noderna inte behöver någon kristall eller keramisk oscillator för att kunna synkronisera sig med bussen. LIN är baserat på Universal Asynchronous Receiver, Transmitter/Serial Communications Interface (UART/SCI) och har en överföringshastighet på 20 Kbps.

LIN-kommunikation använder sig av så kallade ramar, eller *frames*, som består av en header- och en response-del. I headern finns ett 13-bitar långt stycke med dominanta bitar som kallas break, en synkroniseringsdel som är en byte lång och har värdet 55_{16} och en givare som inne-håller en 1 byte adress till mottagarenheten. Detta skickas av den enhet som begär att få data från en annan enhet. Response-delen av en *frame* skickas av den enhet som fått en förfrågan om data, denna del innehåller en till åtta byte data och en checksumma som är den inverterade 8-bits summan med *carry* av alla data-byte.

Varje nod är uppdelad i två delar en ”master task”- och en ”slave task”-del. ”Master task” är den som skickar headern och ”slave task” delen är den som skickar responsen. [10][11][13]

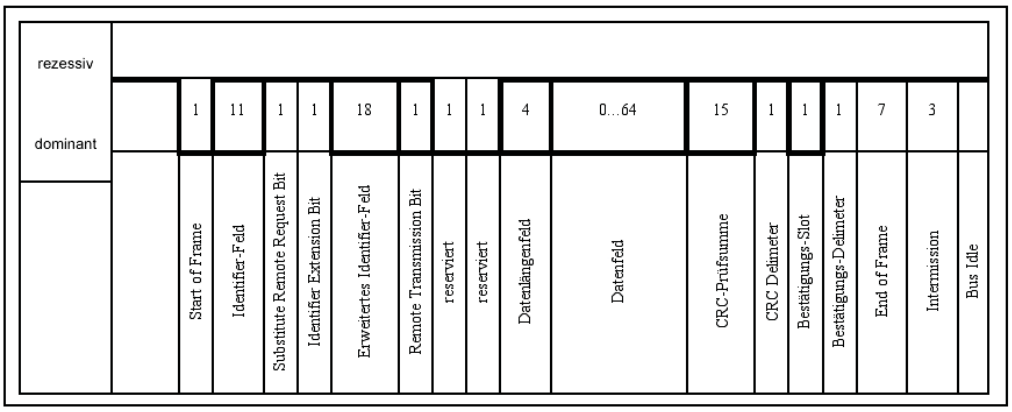
3.4.2 Controller Area Network (CAN)

Till skillnad från LIN-nätverk är ett CAN-nätverk ett 2-trådsnätverk (kan i undantagsfall användas som 1-trådsnätverk) där det kan finnas flera masterenheter, ett så kallat ”broadcast nätverk”. De första CAN-bussarna togs fram under 1980-talet av Robert Bosch.

CAN är meddelande-orienterat vilket betyder att meddelanden skickas när det finns något att skicka och CAN använder så kallad Carrier Sense Multiple Access/Collision Detection (CSMA/DA) teknik. Med *carrier sense* menas att varje nod väntar tills bussen är ledig innan den börjar sända. *Multiple access* betyder att flera noder kan samtidigt försöka använda en buss som inte används. *Collision detection* är sätt att lösa de problem som uppstår då flera noder försöker använda bussen samtidigt. Om två noder försöker skicka ett meddelande samtidigt på bussen måste det finnas ett sätt att avgöra vilken som har högst prioritet. Genom att låta bussen vara normalt hög eller normalt låg ser man till att det finns ett dominant värde. Till exempel om bussen normalt är kopplad hög, genom att den är ansluten till positiv spänning via en kondensator, kommer det att leda till att låga värden är dominanta. När en nod i nätverket skickar ut ett meddelande kommer den första delen i meddelandet vara ett ID för den enhet som skickat värdet. Genom att ange ett ID med bara dominanta bitar kommer den noden alltid att få skicka sitt meddelande först.

Meddelandesäkerheten är hög då det finns felavkänning inbyggt i systemet. Felavkänningen består av ett speciellt fält i varje *frame* som kallas Cyclic Redundancy Check (CRC) som består

av 15 bitar. Deras värde är uträknat genom att använda en polynomekvation. Förhållandet mellan meddelandet och CRC fältets värde är ett känt värde och om detta värde inte stämmer har ett fel inträffat. Den sändande noden kan själv kontrollera om rätt meddelande har skickats på bussen genom att efter meddelandet är sänt kontrollera CRC värdet. Stämmer det inte skickas en *Error frame* och sändaren skickar sedan meddelandet igen.



Figur 4, CAN *dataframe* extended format [3]

Varje *frame* innehåller 0-8 byte med data och överföringshastigheten är upp till 1 Mbit/s vid en kabellängd på 40 meter. Det finns fyra olika typer*; data-*, *remote-*, *error-*, och *overload-frame*. De två trådarna kallas för CANH (HIGH-level CAN voltage) och CANL (LOW-level CAN voltage).

Bosch skrev ingen definition för hur data som skickas med kommunikationsprotokollet skulle vara uppdelad, det vill säga hur varje *frame* ska se ut. Detta gjordes medvetet för att göra CAN mer kraftfullt eftersom varje användare då själv kan välja om det ska satsas mer på exempelvis hastighet istället för överföringssäkerhet. Society of Automotive Engineers (SAE) och International Organization for Standardization (ISO) har dock tagit fram ett antal olika standarder för hur kommunikationen ska se ut och vilken hastighet som ska användas. [12][17][22]

3.4.3 On-Board Diagnostics (OBD)

OBD, eller On-Board Diagnostics, är en standardiserad diagnos-utläsningslösning som används för att läsa ut emissionsrelaterade felkoder. Första generationens OBD togs fram av California Air Resources Board (CARB), en amerikansk myndighet som arbetar med emissionslagkrav för den amerikanska marknaden, och togs i bruk första gången 1988. Systemet är framtaget för att minska utsläppen från fordon och för att uppmärksamma föraren om det sker någon förändring i något av bilens olika system. 1990 bestämdes det, i och med The Clean Air Act Amendments, att alla bilar som är tillverkade 1991 eller senare för US marknaden ska vara utrustade med OBD. Den första generationen OBD övervakade endast ett fåtal av de komponenter som ser till att utsläppen från fordonet är så låga som möjligt. I och med den andra generationens OBD, kallad OBD II, så övervakas de flesta komponenter i fordonet och från 1996 är det obligatoriskt för alla nytillverkade personbilar för US marknaden att vara utrustade med OBD II. [2]

4 Projektspecifikation

Syftet med projektet är att automatisera en verifieringsprocess med hjälp av HIL-systemet. Ett program ska konstrueras som kan utföra test på ECM-enhetens mjukvara genom felinjicering eller simulering av ett händelseförlopp. Detta för att se hur ECM-enheten utför diagnoser och att den genomför specificerade åtgärder. Ett exempel kan vara att om ett fel har diagnostiserats ska ECM-enheten lagra en felkod i felkodsminnet. Därefter ska ECM-enheten skicka ut ett defaultvärde som simulerar just den givare som orsakat felet för att undvika att fler felmeddelanden lagras på grund av att andra diagnoser, som också är beroende av den givaren, erhållit data från givaren med bristande funktionsförmåga.

Efter att programmet har verifierat processen ska en rapport skapas för att användaren ska kunna kontrollera vilka fel som testet har resulterat i, samt vad som gått bra respektive dåligt. Rapporten ska vara detaljerad för att användaren ska få en fingervisning om var felet har uppstått och vad som gått fel, exempelvis så skulle felet kunna sitta i kalibreringen av ECM-enheten.

4.1 Krav

Programmet ska vara så generiskt som möjligt vad det gäller olika mjukvaror i testobjektet. Genom att, i en Excel-fil, välja vilka fel som ska injiceras så ska testet kunna anpassas till olika fall. Programmet ska vara så användarvänligt och så självgående som möjligt. Därför ska användaren vid start ställa in ett antal parametrar och sedan ska programmet exekvera resterande del utan mänsklig interaktion.

För att programmet ska kunna verifiera att rätt felkod sparas i minnet vid en viss felinjicering ska värden över CAN-bussen läsas av och jämföras med de lästa från ECM-enheten via INCA.

Programmet ska vara så stabilt som möjligt, med andra ord så ska programmet inte avsluta mitt i ett test utan fortsätta trots att mindre problem dyker upp. Vid programfel ska programmet skriva ut i rapporten vad som gick fel och därefter gå vidare till nästa del av processen. Det vill säga, programmet ska alltid köra klart sitt test och skriva ut i rapporten vad som gått fel.

4.2 Funktioner

Programmet ska innehålla sådana funktioner som gör att verifieringen utförs korrekt och i enlighet med gällande lagkrav. Det ska finnas funktioner som kontrollerar Malfunction Indicator Lamp (MIL-lampans) funktionalitet så att den, vid test av emissionsrelaterade, felkoder tänds vid rätt tillfälle. Funktioner som simulerar att bilen kör ett specifikt antal körcyklar krävs för att kontrollera så att felkoder får rätt status vid rätt körcykel beroende på vilken marknad den är konstruerad för. Ytterligare funktioner som gör att programmet utför testerna på olika sätt beroende på vad användaren matar in ska finnas, såsom exempelvis vilken marknads lagkrav som ska följas (EU/US). Annan programvara så som ControlDesk och INCA ska styras automatiskt så att användaren inte behöver vara delaktig i så hög grad.

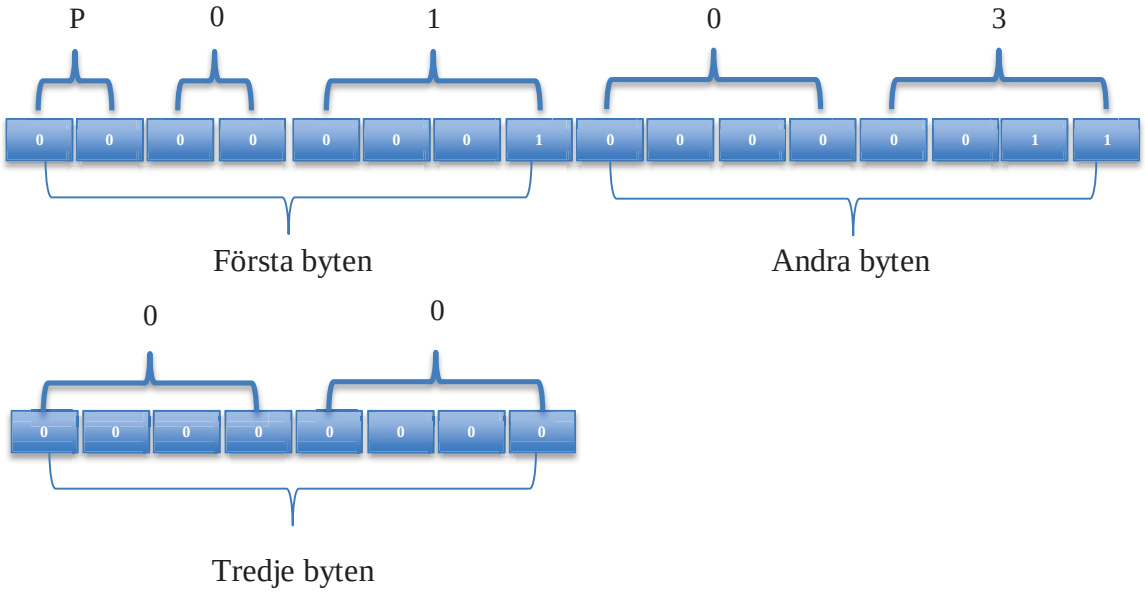
5 Utredning av processer och system

Utredningen i detta projekt består av att reda ut vad ett HIL-system är och hur det är uppbyggt för att med hjälp av det framföra vad som kan göras samt hur lång tid eller hur mycket arbete som krävs för att utföra projekten. Detta ska göras på de processer som getts av Volvo, både felkodsverifieringen och J1699-3 provningen, för att ge inblick i hur och till vad HIL-systemet kan användas till. Utredningen ger i sin tur en översikt över hur omfattande arbetet kan bli, vilka krav som sätts på HIL-systemet för att utföra dessa processer och vilka eventuella vidareutvecklingar som behövs.

5.1 Felkoder

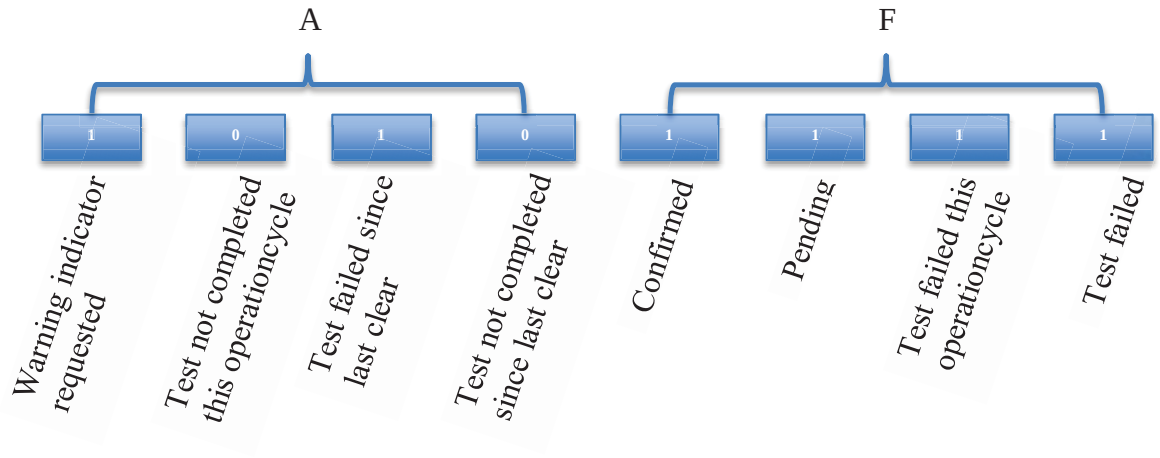
I ECM-enheten finns det två typer av felkoder även kallade DTC (Diagnostic Trouble Code). De två typerna är OBD- samt UDS-koder. OBD-koder är de koder som kan läsas ut med Scan-tool som använder protokoll SAE J1979, utfärdat av SAE, vilket omfattar endast emissions-relaterade felkoder. UDS-koder omfattar alla sorters koder, även emissionsrelaterade, och kan läsas ut med exempelvis DSA som använder protokoll ISO 14229, utfärdat av ISO.

En UDS-kod är uppdelad i tre byte varav den första byten innehåller två bitar som talar om var felet sitter (P=Powertrain, C=Chassi, B=Body, N=Network) och två bitar innehållande den första siffran av DTCn. De resterande 4 bitarna innehåller den andra siffran av DTCn och byte två och tre innehåller de kvarstående siffrorna i DTCn. Detta illustreras i figuren nedan:



Figur 5, Exempel på Powertrain DTC – P010300

En OBD kod har endast de två första byten. Den tredje byten på en UDS kod anger typen av fel, till exempel kortslutning till jord. För varje UDS-kod finns det även en status byte. Status på DTC:n tolkas genom att varje bit kontrolleras enligt figuren nedan:



Figur 6, CAN DTC respons status byte

OBD-koder har bland annat lagkrav för att efter att de har diagnostiserats och status ändrats från *Confirmed* till *Pending* ska MIL-lampan tändas inom maximalt 10 sekunder.[20][23]

5.2 Standarder

För att kommunikationen mellan diagnosverktygen och bilen skall fungera behövs det flera standarder. Här nedan är några av de viktigaste.

ISO 15765 Diagnostics on Controller Area Networks (CAN)

Definierar nätverkslagret, CAN adresser, timing, bithastighet med mera. Reglerar delar av hur själva CAN kommunikationen skall fungera mellan verktyget och bilen. [8]

ISO 14229 Unified Diagnostic Services (UDS)

Alla icke lagkravsstyrda diagnostjänster. Används av Volvo verkstäder. Som till exempel för att läsa ut alla felkoder, även icke lagkravsstyrda. Ta bort felkoder, starta färdiga diagnostester, läsa ut DIDar, programmera immobilizer med mera. [9]

SAE J1979 Diagnostic Test Modes

Lagkravstyrda diagnostjänster. Används av myndigheter och bilprovningen och även av verkstäder. Ta bort felkoder. Läsa ut emissionsrelevanta felkoder, testresultat från vissa diagnoser, VIN numret, mjukvarunumret, mm. Starta läckdiagnos. [16][19]

5.3 Felkodsverifiering

Vid felkodsverifiering kontrolleras det om en felkod fungerar enligt plan och att felkoden följer de lagkrav som finns. Alla felkoder finns som UDS-koder men de lagkravsstyrda felkoderna är de koder som är relaterade till emissioner, så kallade OBD-koder. Dessa felkoder är vad som tänder MIL-lampan för att uppmärksamma föraren att bilen släpper ut mer emissioner än vad lagkravet tillåter. Skillnaden mellan en UDS-kod och en OBD-kod är att UDS-koden är av storleken tre byte och OBD-koden är av storleken två byte.

ECU-enheten söker igenom systemet efter fel minst en gång för varje körcykel, där en körcykel definieras som att bilen startas och motorvarvtalet uppgår till minst 700 varv per minut och bilen stängs av. Detta gäller endast bensindrivna bilar men det lagförda kravet är att motorvarvtalet ska uppgå till minst 150 varv per minut under normal tomgångsvarvtal.

Felkoderna går att läsa ut genom att antingen anropa alla moduler i bilen och fråga om status på alla koder (exempelvis OBD II), eller genom att anropa en specifik modul (exempelvis ECM-enheten). Om en specifik modul anropas är det möjligt att exempelvis fråga om felkoder som gäller just den modulen. Vilket kan vara lämpligt då ett specifikt fel är satt som gäller en viss modul när kontroll ska göras om den förväntade felkoden är satt.

Felkoder har olika status för att ECU-enheten ska ”veta” att ett fel verkligen finns och inte bara är en störning. För att detta ska vara möjligt krävs ett system där alla signaler har specifika kriterier för att ECU-enheten ska uppfatta dem som fel och sedan sätta en felkod. En ECU använder sig av olika status på felkoder för att definiera hur många gånger ett fel upptäckts i bilen och därmed försäkra sig om att det inte bara är störningar. Exempelvis om en felkod ska uppfattas som *Shorten to battery* krävs att spänningen överstiger 16 V över en viss tid. Detta system används för att undvika att oroa bilföraren för små tillfälliga störningar som egentligen inte påverkar bilens funktionalitet.

Det finns många olika status på en felkod. Vid första upptäckten räknar en räknare upp och felkodens status ändras till *Test Failed*. Därefter fortsätter räknaren att räkna upp varje gång felet upptäcks och om felet sitter tills dess att räknaren nått värdet 127 (=felets definition är uppfyllt) ändras felkodens status till *Pending.* Hur mycket räknaren ökar varje gång diagnos-monitorn hittar felet varierar för varje felkod. Det varierar också hur villkoren definieras för när ett fel anses vara ett fel och inte en felmätning. Hur mycket räknaren ökar varje gång ett fel upptäckts beror på dess debounce-klass, men vanligast är att den ökar med 127, vilket medför att status sätts till *Pending* direkt vid första upptäckten.[22]

När felkodens status är *Pending* och samma felkod upptäcks igen vid andra körcykeln ökar räknaren med 127, vilket är vanligast, och felkodens status ändras till *Confirmed*. Detta gäller för en US bil. Är det däremot en EU bil så stannar felkodens status på *Pending* och först vid tredje tillfället ändras felkodens status till *Confirmed* (se tabell nedan). Om felkoden inte upptäcks igen kommer felkoden att raderas från felkodsminnet.

Tabell 1, statustilldelning för DTCer

Körcykel	Status US	Status EU	Antal fel upptäckts
1	Pending	Pending	1
2	Confirmed/Permanent	Pending	2
3	Confirmed/Permanent	Confirmed	3

Först då felkodens status är *Confirmed* och om det är en emissionsrelaterad felkod, tänds MIL-lampan. För att MIL-lampan ska slockna krävs att bilen genomför tre körcykler och att felet inte kan upptäckas, om så är fallet kommer MIL-lampan vid start av den fjärde körcykeln att vara släckt. Dock har inte felkodens status ändrats från *Confirmed*, för att *Confirmed* ska tas bort krävs att bilen kör 40 stycken Warm-Up Cycles (WUC). Felkoderna kan också tas bort med hjälp av ett verktyg, exempelvis Scantool. Kriterierna för en WUC är att motor-temperaturen är under 70°C vid start och att motortemperaturen någon gång under körningen överstiger 70°C, samt att motortemperaturen ska ha ökat minst 22°C under körcykeln.

För att förhindra fusk vid besiktningar har man infört begränsningar så att det inte är möjligt att radera felkoderna fullständigt. Då MIL-lampan tänds så ändras felkodens status till *Confirmed* och felkoden sätts i *permanentfelkodsminnet. Permanentfelkodsminnet* är ett separat non-volatile minne som behåller sitt innehåll även om spänningen försvinner. Detta finns för att fel-koder inte ska kunna raderas genom att koppla ur batteriet. *Permanentfelkodminnet* är därmed omöjligt att rensa, då det enda som kan rensa detta minne är ECU-enheten själv. Detta gör den då den räknat tre helt felfria körcykler efter varandra (alltså då MIL-lampan släcks). Först vid start av den fjärde körcykeln har felkoden tagits bort från *permanentfelkodsminnet*. Detta illustreras i tabellen nedan:

Tabell 2, Återställning av DTCer i *permanentfelkodsminnet*

Antal cykler utan fel upptäckt	Status US	Status EU	Antal gånger fel upptäckts
1	Confirmed/Permanent	Confirmed/Permanent	0
2	Confirmed/Permanent	Confirmed/Permanent	0
3	Confirmed/Permanent	Confirmed/Permanent	0
4	Confirmed	Confirmed	0

Beroende på vad för fel som är satt i bilen finns det skydd som består av *failsafe* och *inhiberingar*. *Failsafe* är till för att bilen ska kunna fortsätta köras utan att den skadas eller får motorstopp om ett fel uppstår. För varje felkod så finns det en så kallad *mask*, där varje *mask* har villkor som talar om för ECU-enheten när en *failsafe* respektive en *inhibering* ska aktiveras och avaktiveras. Ett exempel på en felkod som kräver *failsafe* och *inhiberingar* är *AMBOFF*, som sätts om det blir något fel med den givaren som mäter omgivnings-temperaturen. Utan en *inhibering* kommer alla funktioner som använder sig av denna givare att sätta felkoder. Detta medför att när man läser ut fel-koderna från bilen kan man se att det är ett fel på just *AMBOFF* funktionen, annars hade alla andra funktioner som var beroende av den givaren skickat ut felkoder och möjligheten att då upptäcka vad som var det ursprungliga felet hade varit svårare. En aktivering av en *failsafe* gör att en annan signal ersätter AMBOFF signalen. Andra åtgärder kan vara att man till exempel begränsar motorns prestanda. Den idag manuella processen som görs för att kontrollera fel-koders funktionalitet är relativt tidskrävande och består av följande steg:

Figur 7, Brytbox, som idag används för att injicera fel

1. Rensa felkoder
2. Kontrollera att koder är rensade genom att starta bilen och läsa av felkoderna igen
3. Sätt ett fel, till exempel bryt en kontakt manuellt på brytboxen och lämna kretsen öppen
4. Starta bilen
5. Kontrollera att felkodens status ändrats till *Pending*
6. Stäng av bilen
7. Starta bilen
8. Om US modell, kontrollera att felkodens status är ändrad till *Confirmed* (MIL-lampan tänds)
   Om EU modell, kör en körcykel till och kontrollera därefter status
9. Kontrollera att inhiberingar körts som de ska enligt protokoll
10. Kontrollera att felkodens status är *Confirmed*, om inte testa igen från punkt 1, stäng av bilen. Sätts inte felkoden som *Confirmed* på rätt körcykel, notera felet och gå vidare till nästa felkod.
11. Rätta felet
12. Kör tre körcyklar
13. Kontrollera att felkoden är borttagen från *permanentfelkodsminnet* genom att kontrollera att MIL-lampan slocknat (eller felkodens status är ändrad till *History* om man läser ut koden i INCA)
14. Om allt gått bra så fungerar felkoden som den ska

Notera: Vid varje kontroll måste felkoderna läsas ut med både OBD II (överförings-protokoll) eller DSA och INCA (program).

Felkodsverifieringens krav på program- och hårdvara

För att kunna utföra en verifiering krävs det att programmet kan styra programvarorna INCA och ControlDesk samt att det kan kommunicera över CAN-bussen. Med INCA måste data kunna läsas från felkodsminnet och *permanentfelkodsminnet*. Det måste vara möjligt att rensa felkodsminnet och det måste vara möjligt att läsa MIL-lampans status. I ControlDesk måste det vara möjligt att sätta på och stänga av tändningen, motorn måste gå att starta och stänga av, det ska vara möjligt att sättas ECM-enheten i sleep-mode och sedan väcka ECM-enheten igen och slutligen måste det vara möjligt att injicera och återställa fel i bilen.

5.4 SAE J1699-3 provning

J1699-3 provning används för att kontrollera om OBD-kommunikationssystemet fungerar korrekt så att exempelvis verkstäder kan läsa ut eventuella fel med ett standardiserat verktyg. Det följer SAE J1979 och därför har Volvo identifierat specifika givare för att verktygen ska kunna utläsa informationen ur svaret från en service. Dessa givare kallas för *Parameter Identifiers* (PID) och är adressen till givarna för att identifiera dem för verktygen. OBD-felkoder tänder MIL-lampan i bilen och J1699-3 kontrollerar bland annat att det utförs på ett korrekt sätt. Det är ett standardiserat test som alla biltillverkare måste uppfylla för att få bilen godkänd för att säljas på EU och US marknaden.

I J1699-3 ingår två typer av test, statiskt och dynamiskt. Det statiska testet har fem huvuddelar: testning och utläsning av DTCs (Diagnostic Trouble Code) utan satta felkoder, med status *Pending*, med status *Confirmed*, med felet åtgärdat och efter tre slutförda körcykler.

I det dynamiska testet så testas olika diagnossystem för emissionsrelaterade fel. De diagnoser som ska initieras kräver att föraren utför testet på ett visst sätt, exempelvis att bilen ska ha kört 30 sekunder på tomgång, 300 sekunder i över 40 km/h och att den totala körtiden är 600 sekunder. [18]

Start av J1699-3 (både statiskt och dynamiskt test)

Q1:	Först efterfrågas vilket kommunikationssystem som används för att läsa ut DTC.
Q2:	Därefter efterfrågas om man ska köra ett statiskt eller ett dynamiskt test.
Q3-Q9:	I fråga tre till och med nio ska parametrar rörande vilken bil som testet utförs på anges, parametrarna är: årsmodell, märke, modell, antal OBD-kompatibla ECU-enheter, antal programmerbara OBD ECU-enheter, motortyp (diesel eller bensin) samt typ av drivlina (hybrid, konventionell, start/stop eller plug-in hybrid).
Q10:	Därefter ska vilken standard som ska appliceras anges, följande val är möjliga US, EU, Heavy Duty US eller Heavy Duty EU.
Q11:	Sist ska storleken på bilen anges, Light Duty (personbil) eller Medium Duty (lastbil).[18]

Statiskt test

Det statiska testet består av fem huvuddelar:

1. Först testas bilens funktioner utan att några felkoder är satta (test av MIL-lampa, Service $04, $06, $01, med mera)
2. Därefter testas bilen med en felkod med status *Pending*, detta utförs genom att injicera ett fel manuellt och köra en körcykel (test av Service $07, $03, $02).
3. Sedan testas att felkoden ändrar status till *Confirmed* efter ytterligare körcyklar med felet fortfarande kvar (test av Service $07, $03, $02).
4. Därefter återställs felet, vilket betyder att felet åtgärdas, och under tiden kontrolleras det att MIL-lampan inte släcks och att felkoden finns i *permanentfelkodsminnet* (test av Service $07, $03, $02, $0A).
5. Till sist körs ytterligare tre körcykler för att kontrollera att MIL-lampan släcks och, om så önskas, kontrolleras det att felkoden sätts i och tas bort ur *permanentfelkodsminnet* som den ska (kräver att ett fel injiceras igen och helas) (test av Service $07, $03, $02, $0A).[18]

Dynamiskt test

Det dynamiska testet består av två huvuddelar:

1. Verifiering av In-use performance counters Service $01 och Service $06 (här körs service $09, $01, $04, $01, $06, $03, $09, $01, $09).
2. Verifiering av In-use performance counters Service $01 och I/M Readiness (här körs service $01, $06, $09, $01, $03, $07, $04, $09).[18]

SAE 1979 Protokoll – Service för emissionsrelaterade felkoder

Eftersom J1699-3 är ett test som utförs för att testa lagkravstyrd diagnoskommunikation så kommer endast Service $01-$0A att användas. En service är en tjänst som alla ECU enheter har och de svarar med data, exempelvis om service $01 PID01 frågas ska ECU-enheten svara om PIDen stödjs och med dess senaste uppmätta värde. Nedan finns en översiktlig beskrivning av varje service och dess funktion:

Service$01:
Varje ECU måste svara med hur många PIDer som stödjs och alla PIDer som stödjs måste svara med det senaste uppmätta värdet av systemet, om en negativ respons erhålls så ses det som ett misslyckat test.

Service$02:
Varje ECU måste svara med specifik information som innehåller data om den efterfrågade PIDen för den specifika frågan. Svaret ska erhållas i *freeze frames*, vilket är sparad data vid fel-tillfället. *Freeze framen* innehåller motorns tillstånd (så som motorhastighet, bilens hastighet, motortemperaturen etc.). Därefter skickas även en förfrågan om vilken DTC som orsakat felet.

Service$03:
Varje ECU-enhet ska svara individuellt och svaret ska bestå av ett meddelande som innehåller alla lagrade OBD-koder där varje DTC består av två byte.

Service$04:
Alla ECU-enheter ska radera alla emissionsrelaterade DTCer. Systemet ska endast radera DTCer då tändningen är på och motorn är avstängd, om dessa krav inte uppfylls ska ECU-enheten svara med negativ respons $7F_{16}$, följt av frågan 04_{16} och conditions-not-correct 22_{16}.

Service$06:
För varje ECU ska svaret bestå av de senaste maximala och minimala värdena som erhållits, oberoende av hur många körcyklar som körts. Om inga värden finns (till exempel om batteriet har kopplats ur) så ska svaret bestå av *$0000*. Detta används för att bilmekaniker ska kunna läsa ut de senaste mätvärdena ur bilen. De är utformade så att svaret ges med lämpliga enheter, exempelvis tryck anges i Pascal/Bar.

Service$07:
Svaret innehåller alla felkoder med status *Pending* (i samma form som i *figur 2* i *kapitel 5.1.1*) som inträffat den senaste körcykeln och är till för att en bilmekaniker exempelvis ska kunna kontrollera om felet fortfarande är kvar eller är om det är borttaget ur minnet efter det att felet är åtgärdat. Sitter felet kvar blir felkoden tillslut *Confirmed* och då kan felet även erhållas med hjälp av *service $03*.

Service$08:
Används för att starta läckdiagnos. Som är ett test på att bilens tank system inte läcker ut bensinångor.

Service$09:
Varje ECU ska svara med information om bilens så kallade *infotypes*, så som *Vehicle Identification Number* (VIN), *Calibration ID*, *Checksum Verification Number* (CVN), *In-Use Monitor Performance Ratio* (IUMPR), ECU-namn med mera.

Service$0A:
För varje ECU ska svaret bestå av alla fel som blivit *Confirmed* och inte konstaterats åtgärdade av systemet (det vill säga att svaret ska bestå av DTC med status *Permanent*). [20]

J1699-3 - krav på program- och hårdvara

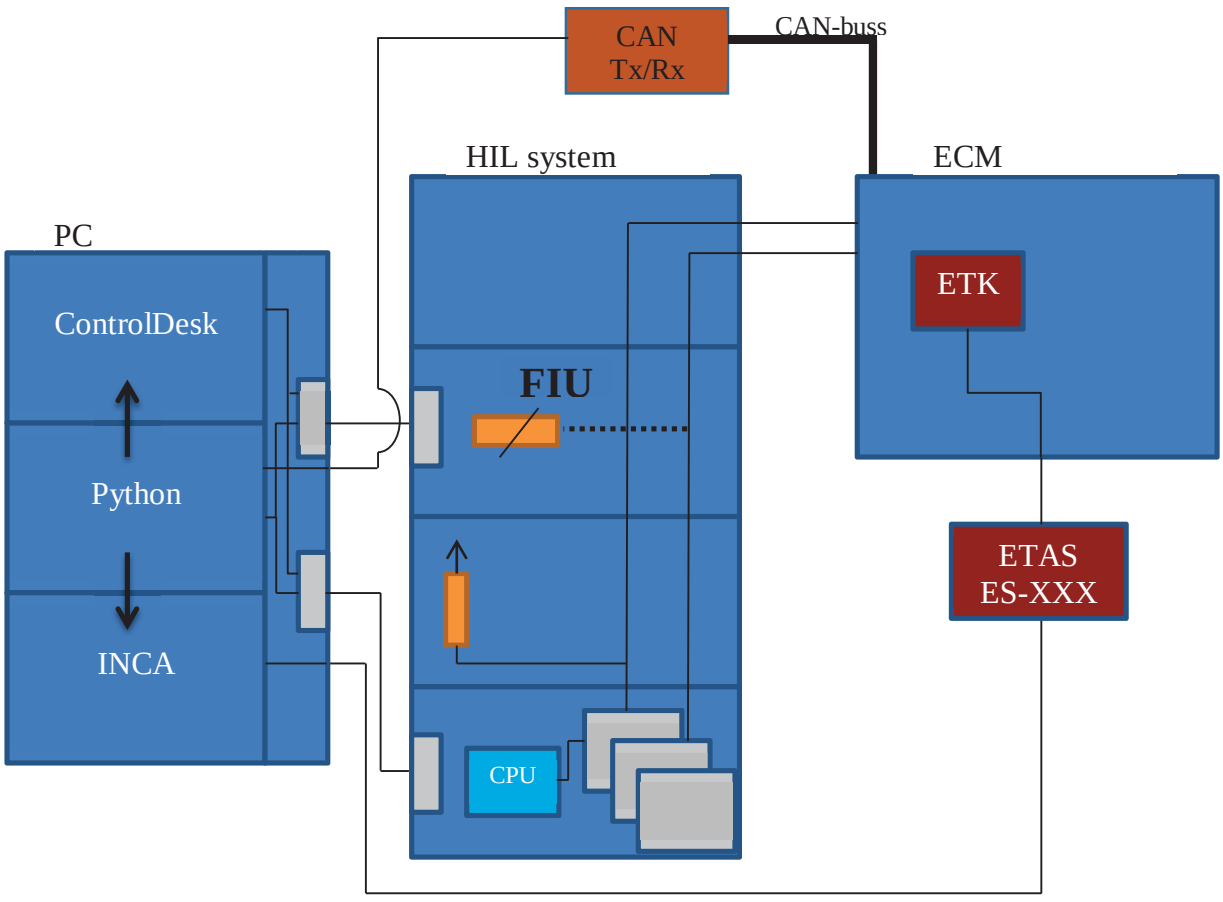
För att utföra J1699-3 provet krävs det att programmet kan kommunicera och styra ControlDesk samt att det kan kommunicera över CAN-bussen. I ControlDesk behövs styrning av start och stopp av motorn, tändning, bilhastighet, motorvarvtal, gas/broms pedal samt möjlighet att sätta ECM-enheten i sleep-mode och sedan väcka den igen. Det måste gå att skicka och ta emot data över CAN-bussen. Detta för att det ska vara möjligt att använda olika Service, så som Service $01-$0A.

5.5 HIL-systemets egenskaper

Testobjektet som ska användas vid verifiering av motorstyrsystemet i bilen är ECM-enheten. När ett HIL-system ska användas för att göra en felkodsverifiering så måste ett fel kunna injiceras och sedan måste det vara möjligt att läsa av vilken felkod som ECM-enheten sätter. För att det ska vara möjligt att injicera ett fel så finns en modul i HIL-systemet som kallas för Fault Injection Unit (FIU). Denna enhet kan injicera ett elfel till de signaler som är kopplade till ECM-enheten. De fel som kan injiceras med hjälp av HIL-systemet och som kommer att användas är följande; öppen krets, kortsluten mot jord och kortsluten mot positiv spänning. Detta realiseras med hjälp av reläer.

För att HIL-systemet ska kunna simulera alla signaler som ECM-enheten behöver kopplas även fysiska komponenter in. På så sätt förhindras att ECU-enheten lagrar oönskade felkoder i systemet till följd av att elfel upptäcks.

För att styra HIL-systemet används programvaran ControlDesk, från dSpace, och för att mäta signaler i ECM-enheten används ett mätverktyg från ETAS som kan styras med hjälp av programvaran INCA. Huvudkommunikationen mellan olika ECU-enheter i bilen sker över CAN-bussar och dessa kan läsas av med en CAN sändare/mottagare. Alla dessa program och enheter kan i sin tur styras med hjälp av programspråket Python.



Figur 8, Översiktsbild HIL-system, PC och ECM

I HIL-systemet körs Simulinkmodeller av en bil, det finns två olika typer modeller, en för dieselmotor och en för bensinmotor. Dessa modeller kan styras via ControlDesk och använder sig av en motorlast som simulerar motorn med hjälp av motstånd och komponenter men den största delen av modellen är mjukvarusimulerad. Det finns även möjlighet att koppla in en transmissionslast för att det ska vara möjligt att koppla en fysisk växelspak. Med hjälp av dessa modeller kan simulering av signaler utföras, exempelvis starta tändning, starta motor, öka motorvarvtalet, ändra motortemperatur och bromsa.

I ControlDesk kan även felinjicering åstadkommas genom att sätta enkla elfel såsom öppen krets, kortslutning mot jord eller kortslutning mot batteri. Detta kan göras med hjälp av reläer inuti FIU-enheten.

På Volvo används programspråket Python när det gäller att styra ControlDesk och andra programvaror. Funktioner som upprättar kommunikation med HIL-systemet finns färdigt på Volvo såväl som styrning av enkla signaler som exempelvis starta och stoppa motorn. Dessa kan med fördel användas då ett exempelprogram ska tas fram.

5.5.1 HIL-systemets brister

HIL-systemets modeller innehåller en del brister för att felkodsverifieringen och J1699-3 provningen ska kunna utföras korrekt. Vid uppstart av en modell finns redan felkoder i minnet vilket medför att det aktiveras en del *failsafe* och *inhiberingar*. Detta orsakar att testobjektet (ECM-enheten) kan uppföra sig på ett oönskat sätt och medföra en osäkerhet i verifieringen så att en del tester misslyckas. Exempelvis skulle en felkod kunna inhiberas på grund av att en annan felkod finns i minnet vilket resulterar i att den förväntade felkoden inte återfinns i minnet vid en felkodsverifiering. Detta medför att programmet som ska konstrueras måste kunna blockera dessa felkoder. Modellerna är dock under utveckling och nya versioner är på gång.

6 Programuppbyggnad

6.1 Val av process

Efter utredning av systemet och dess möjligheter beslutas att automatisering av felkodsverifieringen är genomförbar. Detta baserat på att ett sådant program går att konstruera på ett, för Volvo, tillfredsställande och generiskt sätt och därmed möjliggöra vidareutveckling genom att lägga till ytterligare funktioner.

Med tanke på omfattningen av att automatisera dessa processer har felkodsverifieringen valts. Dels för att mycket av de funktioner som krävs i programmet redan finns tillgängligt och dels för att J1699-3 provning kräver mycket mer arbete i form av kalibrering av svårsimulerade signaler som exempelvis motortemperatur.

6.2 Val av system

Det system som idag finns på Volvo Cars för att simulera en bil är HIL-systemets motormodeller. Dessa styrs med hjälp av en programvaran ControlDesk, som tidigare diskuterats i kapitel 2.2.1. För att kunna mäta direkt i ECM-enheten används programvaran INCA, och för att kommunicera över CAN-bussen används en så kallad XL vector box. Alla dessa programvaror och enheter kan styras med hjälp av programspråket Python och Volvo har sedan tidigare färdiga funktioner skrivna i Python för att upprätta kommunikation med de olika enheterna och programvarorna. Dessa funktioner är dock inte ihopsatta till ett komplett program för att exempelvis kunna utföra verifiering av felkoder.

För att skapa ett enkelt interface för inmatning och utmatning av data valdes Excel som är en programvara som de flesta är vana vid och som används mycket på Volvo. Detta leder även till att programmet blir enkelt att använda eftersom installation av ytterligare programvaror undviks.

6.3 Programdesign

För programmet har sekventiell kodning använts. Detta för att det blir lättare att verifiera att varje steg har gjorts innan nästa steg påbörjas. Programmet designas så att implementering av andra motortyper kan göras enkelt, dock konstrueras detta program endast för bensinmotorer (VEP).

6.3.1 Inmatning av data

Det första data som programmet vill få inmatat är vilken region som ECM-enheten är programmerad för (EU/US). Detta för att det finns vissa skillnader i kraven för de olika områdena, till exempel efter hur många körcykler som felkodens status ändras till *Confirmed*. Om området som ECM-enheten är programmerad för till exempel är US kommer felkodens status ändras till *Confirmed* efter två körcykler där samma fel hittats. Om däremot ECM-enheten är programmerad för EU marknaden så kommer felkoden inte ändra statusen förrän i tredje körcykeln. Då ECM-enheten sätter ett antal felkoder direkt när den kopplas in till HIL-systemet och utan att något fel har injicerats finns i inmatningsmallen en kolumn där felkoder

som ska blockeras när programmet startar kan skrivas in. Detta för att programmet ska vara så användarvänligt som möjligt då användaren annars skulle behöva gå in i koden för programmet och ändra vilka felkoder som ska blockeras. De felkoder som lagras är olika för olika motor-typer (diesel och bensin) men för det konstruerade programmet implementeras endast koder för VEP-motorer. Nedan finns en tabell som visar hur inmatningsmallen ser ut.

Tabell 3, Del av inmatningsmall för VEP

Part/component	Expected fault code	MIL? (Y/N)	Open circuit	Shorten to GND	Shorten to battery	Blocked fault codes
CAN1H_Propulsion		N				L2ACMCS
CAN1L_Propulsion		N				TPSB
LIN 7 - Slave node, TCM is master		N				TPS2H
AC High Pressure		N				TPS1H
MAP sensor intake air		N				CPUDIS
Accelarator pedal PWM signal	APSPWM_H	N				ICDIS
Brake Vacuum Sensor for ISS (313C)	STMPDL	N				GPINIT
Ambient temperature sensor -		N				PWMAFE
Ambient temperature sensor +		N				ATPR
Clutch Pedal Position Sensor		N				CANTCM
Brake pedal light switch (EUCD)		N				GPRECIVE
Camshaft Sensor Exhaust	CAEELEA	Y				CACPAMBP
Brake Diagnostic Switch (C1MCA)		N				UOXM_L_A
Camshaft sensor Intake		N				END OF LIST
Roots compressor bypass valve -		N				
Vacuum Regulator Turbo Wastegate	ARTCVOP,ARTCVGS, ARTCVBS	N				

I inmatningsmallen finns ett antal förifyllda kolumner. Dessa innehåller namn på komponenten, den felkod som förväntas sättas, vilken pinne på ECM-enheten som komponenten är kopplad till och om felkoden är MIL-relaterad eller inte. Det finns även tre kolumner där ett kryss kan sättas för de felkoder som ska köras, varje kolumn motsvarar en viss typ av fel.

Programmet kommer att gå igenom listan en komponent i taget och sätta de fel som ska sättas för den komponenten. Felen kommer att sättas i följande ordning *Open circuit*, *Shorten to GND* och sist *Shorten to BATT*. Vilka av dessa fel som kommer att sättas beror på vilka rutor som är ikryssade. De rutor som är gråmarkerade går inte att köra, detta på grund av hårdvarubegränsningar i HIL-systemet och ECM-enheten. Om ett kryss sitter i en av dessa rutor kommer programmet att skriva ut att felet inte går att sätta och sedan gå vidare till nästa test.

I kolumnen där den förväntade felkoden finns kan flera felkoder skrivas in med ett kommatecken som separator för att ange om det är olika felkoder för de olika feltyperna. Detta kan till exempel ses på raden för komponenten *Turbo Vaccum Regulator Wastegate* i figuren ovan där alla tre typerna av fel kan sättas. De förväntade felkoderna måste alltid sättas i den ordning som kolumnerna är placerade, alltså först *Open Circuit*, sedan *Shorten to GND* och sist *Shorten to BATT*.

6.3.2 Kommunikation med ControlDesk

För att styra HIL-systemet så kommunicerar programmet med ControlDesk. Genom ControlDesk kan variabler styras och läsas av och därmed kan en körcykel initieras. För att en körcykel ska vara korrekt genomförd måste vissa villkor uppfyllas vad det gäller varvtal och att ECM-enheten går ner i sleep-mode mellan varje körcykel.

För att upprätta kontakten med ControlDesk och därmed HIL-systemet så måste ett antal olika portar initieras. Funktioner för att göra detta finns sedan tidigare och dessa implementerades i programmet för att initiera portarna.

För att till exempel kunna starta bilen så måste ett antal olika variabler, som sedan skickas som signaler till HIL-systemet, styras så att ECM-enheten vaknar och tändningen kopplas till. Därefter måste en förfrågan skickas om att starta motorn. Efter att ett varvtal över 700 varv per minut är uppmätt så sätts startförfrågesignalen till låg. För att kunna simulera en körcykel stängs sedan bilen av, ett antal olika mätningar görs och sedan startas bilen igen på samma sätt som tidigare. Mer ingående beskrivning av mätningarna finns i kapitlet *6.3*.8 *Körcykel*.

6.3.3 Kommunikation med INCA

Ett användbart program för felkodsverifiering är INCA, där man kan mäta direkt i kärnan för att ta reda på exempelvis vilka felkoder som är satta och om MIL-lampan är tänd. Precis som för kommunikationen med ControlDesk finns det funktioner sedan tidigare för kommunikationen med INCA och även denna gång har många av dessa använts.

När programmet initierar kommunikationen med INCA initieras ett antal olika portar som är kopplade till ECM-enheten. Med hjälp av INCA kan till exempel minnet där felkoderna finns läsas. Det går att läsa vilken status felkoden har, *Pending* eller *Confirmed*. Det går även att kontrollera om det sitter några felkoder i *permanentfelkodsminnet*. De variabler som läses ur ECM-enheten jämförs sedan med den förväntade felkoden och statusen kontrolleras för att se om felkodens status ändrats till *Confirmed* under rätt körcykel. Därefter kontrolleras att felkoden tagits bort ur *permanentfelkodsminnet* vid rätt körcykel.

För att kontrollera felkodens status läses en minnesplats som innehåller ett 8-bitars ord, där bit två indikerar om felkoden är *Pending* och bit tre indikerar om felkoden är *Confirmed*. Med INCA går det även att läsa ut statusen för MIL-lampan, det vill säga om den är på eller av.

6.3.4 Kommunikation via CAN-buss

För att få möjlighet att läsa ut och skicka data över CAN-bussen används en XL vector box. En XL vector box är en CAN-sändare och -mottagare. På CAN sidan används en RS232 anslutning och på PC sidan används en USB-B anslutning. Boxen översätter CAN kommunikationen till seriell kommunikation. För att det ska vara möjligt att använda boxen från Python måste dll-filen länkas till Python biblioteket för XL vector boxen.

I början av programmet initieras CAN kommunikationen. Genom att skicka en förfrågan över CAN-bussen om vilka felkoder som finns i minnet kan man läsa ut vilka koder som har status *Pending* och vilka som har status *Confirmed*.

Vid en förfrågan svarar ECM-enheten med en datasträng som måste tolkas så att namn och status på felkoden kan jämföras med data från INCA och med den förväntade felkoden från inmatningsmallen. Data strängen består av en index byte och tre byte data. Data byten är uppdelade enligt följande, de två första byten är den så kallade P-koden som anger vilken felkod som finns i minnet och den sista byten innehåller information om felkodens status. För att översätta datasträngen så används ett färdigt bibliotek som maskar ut informationen från datasträngen och presenterar informationen som textsträngar.

6.3.5 Skapa en rapport

För att presentera det som verifieringsprocessen har resulterat i produceras på slutet en rapport i Excel format. I rapporten finns information om vilken användare som har startat testet, när testet är gjort och vilken programvara som fanns i ECM-enheten vid testtillfället. Även information om vilka tester som har gjorts och om det finns några anmärkningar skrivs in i rapporten.

För att det ska vara möjligt att kontrollera att rätt felkod är satt i både INCA och på CAN-bussen finns det en kolumn för varje kommunikationsprotokoll. Även information om huruvida status (*Pending* och *Confirmed*) har ändrats vid rätt tillfälle i ECM-enheten och om motsvarande information är läst över CAN-bussen.

I programmet finns en funktion som genererar en notering om det inträffat något som inte ska ske eller om allt har gått som det ska. Det kan vara allt från att ett av kommunikationsprotokollen inte fungerar, att en felkod inte har hittats eller att en status inte har ändrats i rätt körcykel. För exempel på rapporter från programmet se bilaga.

6.3.6 Omvandling av data mellan olika kommunikationsprotokoll och programvaror

Den förväntade felkoden som läses in från inmatningsfilen är en textsträng och för att den ska gå att jämföra med felkoderna inlästa från INCA och CAN-bussen måste ett översättningsscript skapas. När felkoden läses med hjälp av INCA representeras felkoden av *flyttal* och när felkoden läses över CAN-bussen så erhålls en sträng med hexadecimala tal som motsvarar P-koden. För att lösa detta används en Excel-fil där felkoden finns som textsträng, som *flyttal* och som P-kod. Allt detta sparas därefter i listor så att de blir sökbara, vilket möjliggör att programmet, via de olika kommunikationsprotokollen, kan översätta mellan de olika felkodsformaten och därmed kontrollera att rätt felkod är satt.

Korrektheten i de inlästa värdena blir då beroende av hur korrekt Excel-filen är. För att lösa översättningsproblemet finns ett antal olika lösningar med olika filer att läsa in. Ett stort problem är att Excelfilen måste vara uppdaterad för att översättningen ska bli korrekt. För att göra detta problem så litet som möjligt valdes en lösning med så få inmatningsfiler som möjligt.

Dock finns det ett problem med att använda den fil som för närvarande används i programmet. Det finns nämligen inget bestämt utseende på filen så när en annan mjukvaruversion i ECM-enheten ska testas kan det vara så att filen inte går att läsa in. Detta är en stor svaghet i programmet om man vill göra en felkodsverifiering för en annan mjukvaruversion. För att lösa detta problem borde en standardiserad mall för hur dessa filer ska se ut skapas.

6.3.7 Verifiering och jämförelse av data

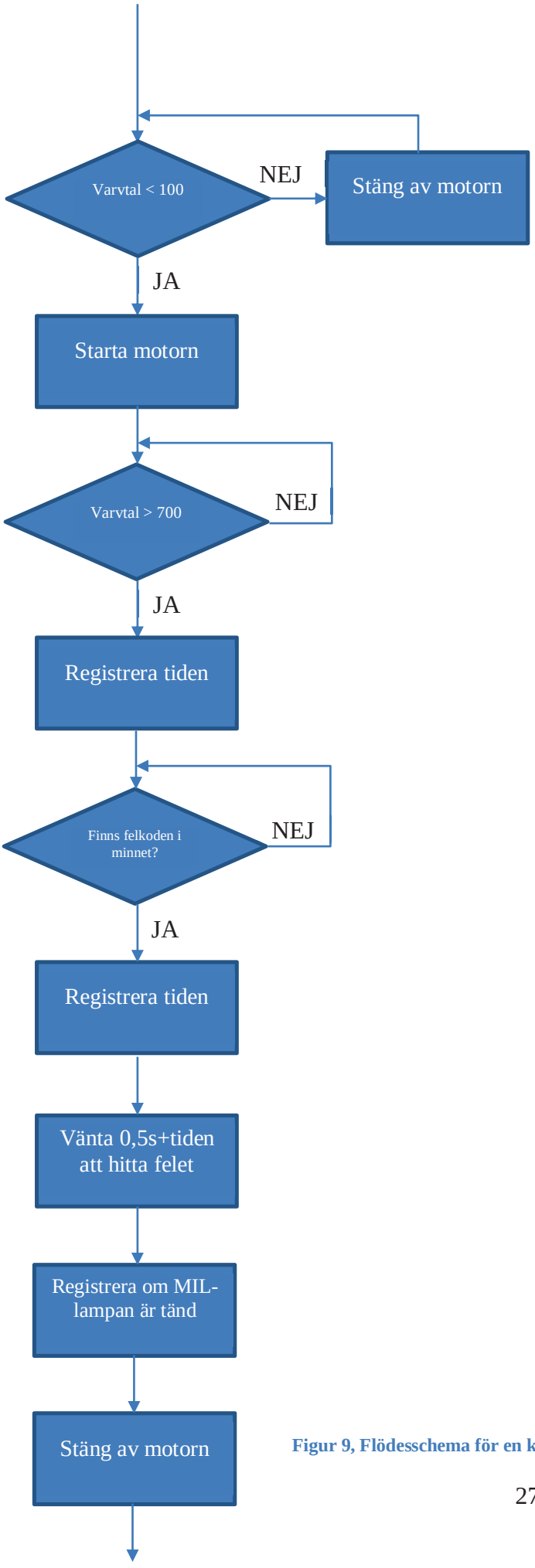
För att det ska vara möjligt att verifiera om rätt felkod är läst med INCA och över CAN-bussen måste översättningsscriptet användas. I programmet översätts värdet från INCA och över CAN-bussen till den form som felkoden är skriven på i inmatningsfilen, vilket i detta fall är en textsträng.

För att kontrollera att de olika felkodernas status har ändrats vid korrekt körcykel loggas det vid vilken körcykel statusförändringen först blivit registrerad och sedan jämförs detta värde med den förväntade körcykeln. Samma sak görs även för kontroll av MIL-lampan samt *permanentfelkodsminnet*.

6.3.8 Körcykel

För att en felkod ska byta status från *Pending* till *Confirmed* måste ett antal körcykler genomföras. En körcykel definieras enligt figuren. Innan motorn startas kontrollerar programmet att motorn är avstängd. Är den igång stängs den av. När programmet är säkert på att motorn är avstängd startas motorn. Efter att motorn startat så sparas tiden och därefter kommer felkodsminnet att sökas igenom tills dess att den förväntade felkoden har hittats. Har inte koden hittats inom tio sekunder kommer programmet att sluta leta då lagkravet för hur lång tid det får gå innan MIL-lampan tänds efter det att felet har diagnostiserats. Efter att programmet sökt efter felkoden väntar programmet en halv sekund plus den tid det tog att hitta felet första gången.

Tiden som programmet väntar efter det hittat felet är satt till 0,5 sekunder plus den tid det tog att hitta felet första gången för att det ska finnas möjlighet för diagnosmonitorn att göra en kontroll innan statusen av MIL-lampan görs och innan motorn stängs av. Därefter kontrollerar programmet om MIL-lampan är tänd med hjälp av INCA och sist kommer motorn att stängas av. Tiden mäts ifrån den tidpunkt då motorn startat. Det är egentligen inte säkert att det är den tidpunkt då diagnos-monitorn hittat felet men för elfel, som är det fel programmet gör verifiering för, hittar diagnosmonitorn felet så fort att den tid det tar att hitta felet är försumbar.



Figur 9, Flödesschema för en körcykel

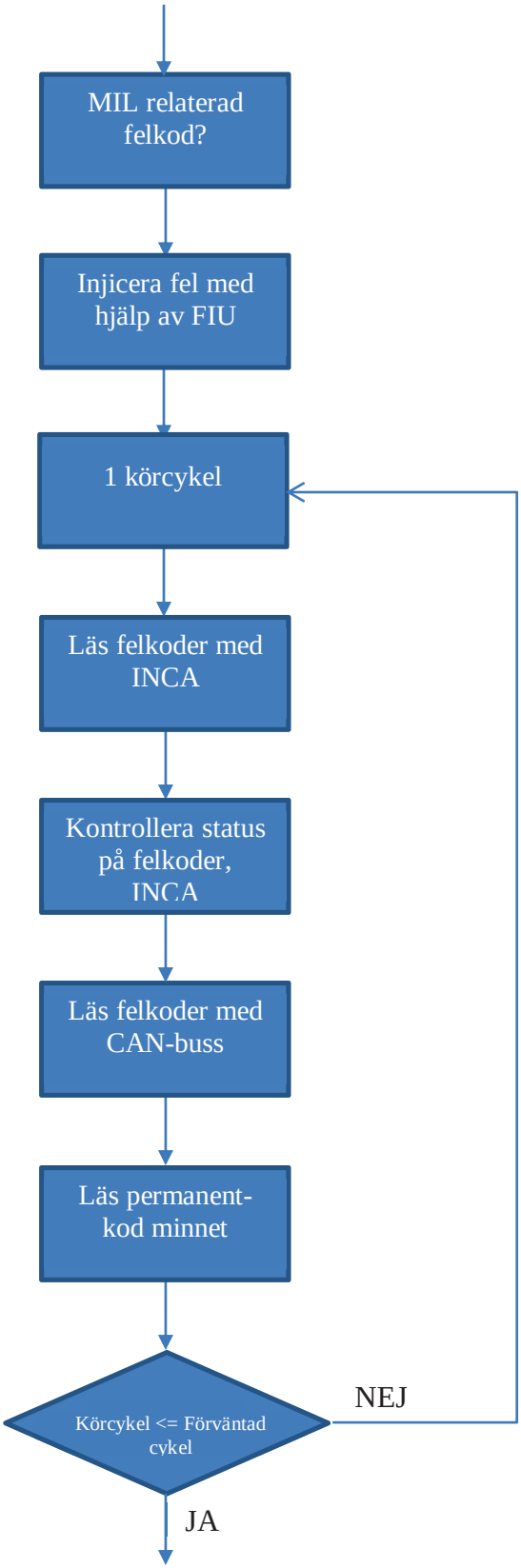
6.3.9 Att injicera en felkod

Det första som kontrolleras när ett nytt fel ska injiceras är om felet är MIL-relaterat, det vill säga om det är en OBD-felkod och därmed om MIL-lampan ska tändas eller inte. Därefter injiceras felet med hjälp av FIU-enheten.

Felet som FIU-enheten sätter kan vara *Open circuit*, *Shorten to GND* eller *Shorten to BATT*. Efter att ett fel är injicerat så körs en körcykel. När körcykeln är gjord används INCA för att läsa av vilka felkoder som finns i felkodsminnet. Vilken status de har kontrolleras genom att maska ut de bitar som anger om koden är *Pending* eller *Confirmed*.

Därefter skickas en förfrågan över CAN-bussen om vilka koder som har status *Pending* respektive *Confirmed*. För att det ska vara möjligt att presentera i rapporten vilken körcykel som felkodens status ändrades och när felkoden sattes i *permanentkodsminnet* läses felkodernas status och vilka felkoder som finns i *permanentfelkodsminnet* varje körcykel.

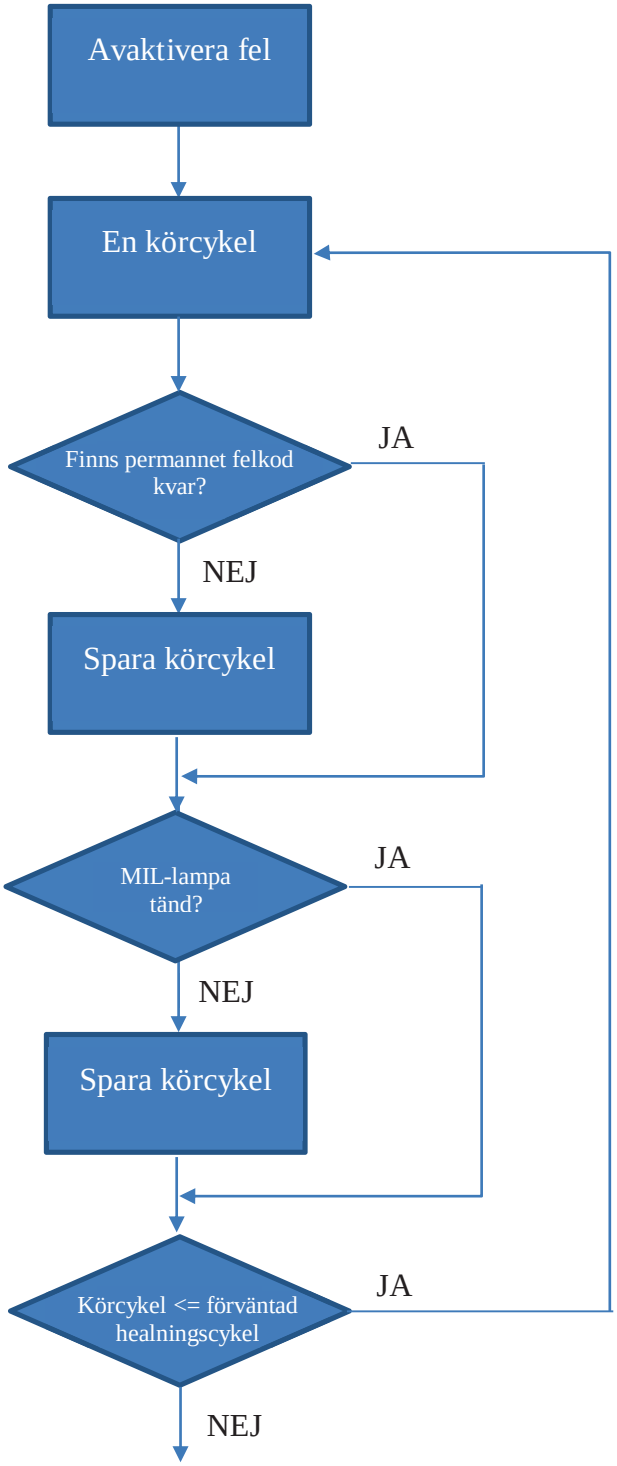
Det sista som kontrolleras är om rätt antal körcyklar är genomförda eller om ytterligare körcykler behöver genomföras.



Figur 10, Flödesschema för att injicera ett fel

6.3.10 Återställning av felkod

För att återställa ett fel så gör programmet enligt flödesschemat i figuren. Först avaktiveras felet, vilket betyder att man tar bort felet genom att återställa och avaktivera FIU-enheten. Sedan genomförs en körcykel där bland annat MIL-lampan och felkodsminnet kontrolleras. Efter det kontrolleras om felkoden fortfarande finns kvar i *permanentfelkodsminnet*. Då felkoden är borttagen sparas vilken körcykel som felkoden försvann från *permanentfelkodsminnet*. För att återställa felet måste ett bestämt antal körcykler genomföras. Antalet är beroende av vilken marknad mjukvaran är skapad för. När programmet har genomfört det förutbestämda antal körcykler för att återställa felkoden kommer programmet att gå vidare.



Figur 11, Flödesschema för återställning av felkod

6.3.11 Felhantering

Om det skulle ske ett fel vid initieringen av ett kommunikationsprotokoll frågar programmet om det ska fortsätta eller om det ska avsluta. Om däremot kommunikationen med både INCA och med CAN-bussen inte går att initiera kommer programmet att avslutas automatiskt. Detta då det inte är möjligt att läsa ut några felkoder utan varken CAN-buss eller INCA-kommunikation.

Om kommunikation med HIL-systemet inte är möjlig eller om det inte går att öppna en kommunikationskanal med FIU-enheten kommer programmet att stoppas automatiskt. Detta då det inte är möjligt att injicera några fel och därmed går det inte att få ut några felkoder.

När programmet avslutas automatiskt kommer programmet ändå skapa en rapport och skriva in en notering i slutet där det specificeras mer exakt vad som orsakat att programmet avslutats.

6.3.12 Huvudprogram

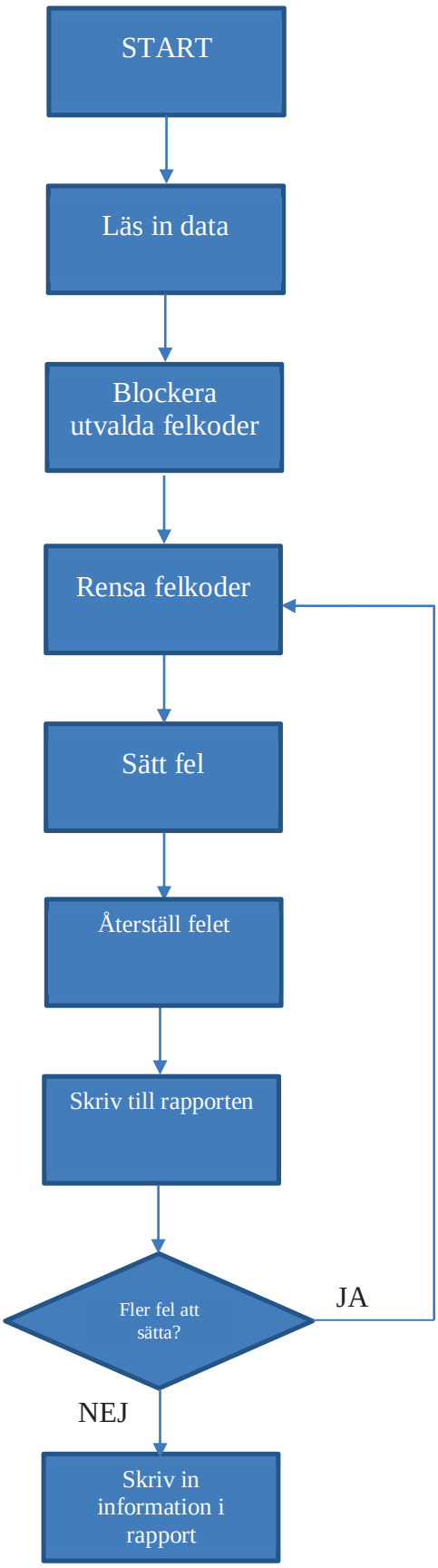
Programmet börjar med att läsa in data från inmatningsmallen och från användaren, dessa data innefattar bland annat typ av fordon och vilken mjukvara som finns i ECM-enheten. Därefter blockeras även de felkoder som är specificerade i inmatningsfilen och felkodsminnet rensas genom att skriva till en variabel med hjälp av INCA. Detta görs för att felkodsminnet ska vara tomt när felet injiceras.

När felkoderna är blockerade injiceras felet, ett antal körcykler körs och felkoderna läses ur minnet. När det specificerade antalet körcykler som krävs för att felkoden ska få statusen *Confirmed* är genomförda återställs felet.

Vid återställningen av felet så kontrolleras *permanentfelkodsminnet* för att se när felkoden försvinner ur minnet och därmed är felkoden återställd.

Under tiden som fel är injicerat och när felkoden återställs kontrolleras MIL-lampans status. När återställningen av felet är klart så skrivs insamlad data till rapporten tillsammans med eventuella noteringar om komplikationer.

När alla efterfrågade felkodstester är genomförda och om ett problem som påverkat alla tester uppstått, kommer en eventuell not skrivas till rapporten. Tillslut kommer rapporten att sparas och programmet därefter avslutas.



Figur 12, Övergripande flödesschema för programmet

7 Resultat

Volvo Cars har gett i uppdrag att utreda i hur stor utsträckning ett HIL-system kan användas för att automatisera olika, idag manuella, processer. De processer som utreds är felkodsverifiering och J1699-3 provning, där ett program som utför felkodverifieringen helt automatiskt kommer att konstrueras.

För att det ska vara möjligt att genomföra en verifiering av felkoder med ett HIL-system krävs en simulerad motormodell där det är möjligt att koppla in en Engine Control Module (ECM). På Volvo Cars finns det sedan tidigare en färdigt konstruerad motormodell för HIL-systemet. Dock sätter ECM enheten ett antal felkoder redan vid uppstart med denna modell. En ny modell är under utveckling och denna modell kommer leda till att antalet på förhand satta felkoder kommer att reduceras signifikant. För att simulera motorn används ett HIL-system från dSpace. HIL-systemet är i sin tur kopplad till testobjektet (ECM-enheten). I HIL-systemet finns en FIU-enhet som kan injicera fel till ECM-enheten och HIL-systemet kan styras med hjälp av programvaran ControlDesk. Med hjälp av programvaran INCA finns det även möjlighet att läsa ut information direkt ur ECM-enheten för att exempelvis kontrollera vilka felkoder som finns i minnet.

Felkoderna som sätts i minnet då ett fel i systemet har upptäcks kan uppnå olika status. De två viktigaste i detta projekt är *Pending* och *Confirmed*. De flesta felkoder uppnår status *Pending* direkt, däremot måste ett antal körcykler genomföras för att de ska uppnå status *Confirmed*. Definitionen av en körcykel är att tändningen slås till, motorn startas, varvtalet överstiger 700 varv per minut (gäller endast bensinmotorer) och motorn stängs av.

Antalet körcykler som måste genomföras beror på vilken marknad som mjukvaran är avsedd för, EU eller US. I fordon för US marknaden kommer statusen för felkoden att ändras till *Confirmed* när felet hittats efter två körcykler och i fordon för EU marknaden kommer detta att göras först efter tre körcykler. Samtidigt som felkodens status ändras till *Confirmed*, och om felkoden är en OBD-kod, kommer felkoden att lagras i *permanentfelkodsminnet* och *Malfunction Indicator Lamp (*MIL-lampan) kommer att tändas. Detta minne kan bara ECM-enheten själv ändra. Detta för att det inte ska gå att radera OBD-felkoder utan att reparera det som orsakade felet. För att återställa en felkod och radera den från *permanentfelkodsminnet* måste tre körcykler genomföras utan att ECM-enheten hittar det fel som orsakade felkoden. Efter dessa tre körcykler kommer även MIL-lampan att släckas.

J1699-3 provningen utförs idag med ett program där användaren måste koppla in datorn till bilens CAN-buss och sedan följa instruktionerna som programmet ger. Instruktionerna innefattar bland annat hur länge bilen ska framföras i en hastighet som överstiger 40 km/h och hur länge bilen ska köras på tomgång. För att det ska vara möjligt att genomföra J1699-3 provningen med hjälp av HIL-systemet så måste det vara möjligt att läsa av och manipulera signaler som motortemperaturen, varvtal och hastigheten, vilket är möjligt i den nuvarande modellen. Vid verifieringen måste strikta protokoll och regler följas för att det ska vara möjligt att genomföra fullständigt. Därutöver, om ett program ska konstrueras för att utföra en J1699-3 provning krävs ett program som kan utföra allt som det nuvarande J1699-3 provningsprogrammet utför, samt att det måste vara möjligt för programmet att kontrollera bilens hastighet, varvtal och motortemperatur. Ett nytt J1699-3 verifieringsprogram måste utföra samma sak som det nuvarande programmet som används i den mestadels manuella J1699-3 verifieringen plus att det måste vara kapabelt att styra HIL-simulatorn på det sätt som idag görs utav den person som idag utför verifieringen. För att det ska vara möjligt att läsa variabler direkt från ECM-

enheten används speciella ECM-enheter som har en ETK-modul inbyggd. Denna modul möjliggör användning av programvaran INCA för att läsa variablerna.

Utredningen ledde till att programmet skrevs med hjälp av programspråket Python. Detta då det är det programspråk som har använts tidigare för styrning av HIL-systemen, kommunikation med INCA och för att läsa CAN-bussen. Därutöver har dSpace implementerat kompabilitet i sin programvara, ControlDesk, för att ge möjlighet att styra HIL-systemet med Python.

För att komma förbi problemet med de vid uppstart redan lagrade felkoderna (som finns på grund av brister i motormodellen) konstruerades programmet med möjlighet till att läsa in data från en Excel-fil där användaren kan specificera vilka felkoder som ska testas samt vilka felkoder som ska blockeras. För att programmet ska kunna genomföra körcykler, samt för att injicera fel med hjälp av FIU-enheten i HIL-systemet kommunicerar det med ControlDesk.

Programmet kommunicerar med INCA för att läsa variabler och felkoder och det läser även felkoder via CAN-bussen. När ett antal körcykler är genomförda med ett fel injicerat och felkodernas status är avlästa återställs felet och ett antal nya körcykler genomförs. Mellan varje körcykel, både när felet är injicerat och när felet inte är injicerat, kontrolleras MIL-lampans status. Mellan varje körcykel kontrolleras det också med hjälp av INCA att felkoderna hamnar i *permanentfelkodsminnet* vid rätt körcykel och att de sedan plockas bort från minnet vid rätt körcykel.

Tiden det tar för en felkod att få statusen *Pending* från det att motorn startats mäts då det finns lagkrav på att det får ta maximalt 10 sekunder från det att felet är diagnostiserat tills det att MIL-lampan tänds. Denna tid används sedan för att definiera väntetiden mellan att motorn startat tills kontrollen av MIL-lampans status och motorns avstängning utförs. Den tid det tar för programmet att hitta en felkod som redan finns i minnet med hjälp av INCA är ungefär fem millisekunder som i detta sammanhang kan räknas som försumbart då det allt som oftast tar minst en sekund för felkoden att sättas i minnet från det att motorn startat.

När alla de felkoder som är specificerade i inmatningsmallen har testats sparas informationen i en rapport i Excel-format. I rapporten anges om och när den förväntade felkoden hittades i minnet med både INCA och CAN-bussen, om de förväntade statusförändringarna har inträffat vid rätt körcykel, om MIL-lampan har tänts vid rätt körcykel och om felkoden har satts och tagits bort ur *permanentfelkodsminnet*.

Då felkoderna läses ur minnet med både INCA och CAN-bussen blir resultatet verifierat med två olika kommunikationsprotokoll vilket medför att det ger en fingervisning om var felet kan befinna sig i systemet. Om värdet från INCA och värdet från CAN-bussen inte stämmer överens och ett av dem är rätt går det att se om det är fel på CAN-bussen.

Programmets funktion har testats genom flera tester där felkodsverifieringar har genomförts. För att kontrollera att programmet fungerar som det ska då kommunikationen med till exempel INCA inte fungerar har INCA stängts ner så programmet inte kan öppna en port till programvaran. För att testa att programmet anger att felkoden inte hittats i felkodsminnet om den inte finns där har fel förväntad felkod angetts i inmatnings-filen. Dessutom har kontroll av att programmet kan hitta den förväntade felkoden i minnet trots att det sitter flera felkoder testats genom att inte alla de felkoder som sitter vid uppstart blockerats. För exempel på rapporter genererade av programmet vid olika förutsättningar se bilagor.

8 Slutsats

Utredningen som gjordes ledde fram till att det är möjligt att genomföra både felkodsverifiering vid elfel och J1699-3 verifiering med hjälp av HIL-systemet. Dock krävs det lite mer arbete för att J1699-3 verifieringen ska vara möjlig att genomföra då kraven är högre för denna verifiering. Ett problem som har upptäckts när det gäller att genomföra verifiering av felkoder med hjälp av HIL-systemet och i synnerhet när det gäller J1699-3 verifieringen, är att det finns en del felkoder som sätts i ECM-enhetens minne vid uppstart av HIL-systemet. Detta beror på att den HIL-modell av motorn som finns för närvarande inte klarar av att simulera alla komponenter på ett tillfredsställande sätt. Nya modeller är dock under utveckling och förhoppningen är att de nya modellerna ska lösa detta problem. För tillfället har detta problem lösts genom att blockera de felkoder som sätts när HIL-system startas upp. Denna lösning fungerar för felkodsverifieringen för elfel men är dock inte optimal eftersom vissa blockeringar kan orsaka störningar i andra verifieringsprocesser.

Det program som skapats för verifiering av felkoder för elfel fungerar som planerat dock finns det ett antal förbättringar som kan göras. Inmatningsmallen där användaren av programmet anger vilka felkoder som ska testas måste uppdateras manuellt då förändringar sker i vilka felkoder som finns och vilka fel som är möjliga att injicera för varje komponent. För att komma förbi problematiken med detta bör programmet antingen själv skapa inmatningsmallen vid uppstart med aktuella komponenter och fel, eller så bör det finnas en funktion i programmet som går att köra vid behov, vilken genererar en ny uppdaterad inmatningsmall. Detta är dock inte möjligt i dagsläget då sådana databaser med kontinuerligt uppdaterad information inte finns än.

När programmet läser felkoderna via INCA, CAN-bussen och inmatningsfilen är de i olika format. För att lösa detta problem var det nödvändigt att en översättningsfunktion skapades. Översättningsfunktionen läser in vilka värden som är länkade till varandra från en fil och därefter är det möjligt att söka i listor, där värdena lagrats, efter vilka värden som hör ihop. Problemet med att använda denna lösning är att programmet är beroende av att den fil som används vid översättningen är uppdaterad och att utseendet på filen alltid är samma. På grund av att översättningen mellan felkodsformaten är olika för olika mjukvaruversioner kan programmet i dagsläget endast utföra felkodsverifiering för elfel på en mjukvaruversion. För att göra en verifiering på en annan mjukvaruversion måste användaren ändra i koden vilken fil som ska läsas in av översättningsfunktionen. Användaren måste också ange hur filen är upplagd när det gäller förhållandet mellan kolumner.

För att få en lösning krävs att det finns möjlighet att läsa ut information från de filer som tillhör den mjukvara som är laddad i ECM-enheten. För att erhålla dessa filer krävs dock ytterliggare programvara samt en funktion som kan utläsa information ur dessa filer.

Ytterligare utveckling av programmet som kan göras är kontroll av *failsafe* och *inhiberingar*. När en felkod sätts i felkodsminnet görs ibland ett antal *inhiberingar* vilket innebär att ett antal andra funktioner i ECM-enheten blockeras som använder sig av samma givare för att de ska sätta felkoder. Detta för att det ska vara möjligt att hitta vad som var ursprungsfelet istället för att minnet fylls av felkoder som satts på grund av följdfel till det ursprungliga felet. När en felkod sätts bör programmet också kontrollera att rätt *inhiberingar* gjorts och om *failsafe* ska aktiveras bör programmet kontrollera att den har aktiverats. Detta har inte implementerats då det är mycket tidskrävande eftersom ett antal filer måste läsas där det är specificerat vilka felkoder som ska blockeras när varje felkod sätts i minnet. Dessa filer har inte heller ett bestämt utseende vilket gör att det är svårt att skapa en funktion som automatiskt läser in dem.

När olika ECM-enheter med olika mjukvaruversioner ska verifieras måste också olika sdf-filer och sddb-filer användas. I det nuvarande programmet behöver användaren ändra i koden för att ändra vilka filer som används för detta. Vid utveckling av programmet bör programmet av sig självt plocka fram rätt filer för just den mjukvaruversionen.

Innan programmet startas måste användaren själv starta ControlDesk och ladda in rätt sdf-fil i HIL-systemet då den sdf-fil som laddas in i ControlDesk måste vara samma som den som verifieringsprogrammet använder. Användaren måste också själv starta ett experiment i INCA och ladda in den mjukvaruversion i ECM-enheten som ska testas. En vidareutveckling av programmet som skulle underlätta för användaren avsevärt är om programmet också kunde göra detta själv utifrån vilken mjukvaruversion som användaren vill testa.

Referenser

[1] Anonymous, "ETAS GmbH," *European Automotive Design*, vol. 4, p. 27, Aug 2000

[2] A. R. B. California Environmental Protection Agency. (2015, 15 April). *On-Board Diagnostic II (OBD II) Systems - Fact Sheet*. Available: http://www.arb.ca.gov/msprog/obdfaq.htm

[3] D. Fiedler, "CAN-Data Frame im Extended Format," vol. 873 × 351, E. can-dataframe, Ed., ed. Wikimedia commons: Abwandlung von Denis Fiedlers Grafik, 2007, p. CAN data frame extended format.

[4] dSpace, "ControlDesk Next Generation," vol. 1011 x 703, KeyVisual_CDNG_5.0_Compact, Ed., ed. dspace.com: dSpace, 2015, p. Universal experiment software for electronic control unit (ECU) development.

[5] dSpace, "dSPACE Simulator Full-Size," vol. 681x755, FullSize-Simulator_130420_vv4, Ed., ed. dSpace.com: dSpace, 2015, p. Picture of a dSpace full size HIL.

[6] L. Eriksson and L. Nielsen, *Automotive : Modeling and Control of Engines and Drivelines*. Somerset, NJ, USA: John Wiley & Sons, Incorporated, 2014.

[7] ETAS. (2015, June 1). *INCA base product*. Available: http://www.etas.com/en/products/inca-details.php

[8] I. O. o. Standardization, "Road vehicles - Diagnostics on Controller Area Network (CAN)," in *Requirements for emissions-related systems*, ed: ISO, 2003.

[9] I. O. o. Standardization, "ISO 14229-1," in *Road vehicles - Unified diagnostic services (UDS)*, ed: International Organization of Standardization, 2011.

[10] Z. Kaspar and J. Kuhn, "Automotive Networking: An Introduction to LIN," *ECN*, vol. 50, pp. 27-28, Mar 2006

[11] W. Kimberley. (2004) LIN networks in the automotive industry. *Automotive Engineer*. 4.

[12] S. Knutsson, "CAN - Controller Area Network," in *Nätverk i fordon*, ed. Chalmers tekniska högskola: Signaler och system, 2014.

[13] S. Knutsson, "LIN - Local Area Network," in *Nätverk i fordon*, ed. Chalmers tekniska högskola: Signaler och system, 2014.

[14] J. A. Ledin, "Hardware-in-the-loop simulation," *Embedded Systems Programming*, vol. 12, pp. 42-60, Feb 1999

[15] M. McGrath and I. Books24x, *Python*. Leamington Spa, Warwickshire, U.K: In Easy Steps, 2014.

[16] T. Nash, "AUTOMOTIVE PROTOCOLS & STANDARDS," *Motor*, vol. 204, pp. 32-34,36,38-39, Jul 2005

[17] B. Ranjan and M. Gupta, "Automobile Control System using Controller Area Network," *International Journal of Computer Applications,* vol. 67, 2013

[18] S. International, "Surface Vehicle Recommended Practice," in *Vehicle OBD II Compliance Test Cases*, ed. USA, 2012.

[19] S. International, "Surface Vehicle Standard," in *J1979-DA, Digital Annex of E/E Diagnostic Test Modes*, ed. USA: SAE International, 2014, p. 162.

[20] S. International, "Surface Vehicle Standard," in *E/E Diagnostic Test Modes*, ed. USA: SAE International, 2014, p. 164.

[21] M. Short and M. J. Pont, "Hardware in the loop simulation of embedded automotive control system," in *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, 2005, pp. 426-431.

[22] B. Thompson, "UNDERSTANDING CONTROLLER AREA NETWORKS," *Motor,* vol. 209, pp. 46-50,52,54, Jan 2008

[23] V. Cars, "Diagnostic core overview - VEA SPA," vol. 1, ed: Volvo Cars, 2014, p. 113.

Tabell 4, Exempelrapport - för två felkoder finns inte den förväntade felkoden i felkodsminnet och för två felkoder har allting fungerat som planerat. Tiden från start av motorn tills det att felkoden hittades kan ses i kolumnen ”Notes”

Fault code	Component	UDS_DTC	Expected found? (INCA)	Expected found? (CAN)	Pending? (INCA)	Pending? (CAN)	Confirmed? (INCA)	Confirmed? (CAN)	MIL OK? (INCA)	MIL turned off?	Fault code healed?	Fault codes found	Notes
	Change over valve		Not found	Not found	-	-	-	-	N/A	N/A	N/A		, Pending was not set on CAN-bus, Pending was not set in INCA
ARIGT2OP	Ignition signal #2	035213	OK	OK	OK	OK	OK	OK	OK	OK	OK	ARIGT2OP	Time until pending: 2.41s, Completed successfully without complications
ARIGT2GS	Ignition signal #2	035211	OK	OK	OK	OK	OK	OK	OK	OK	OK	ARIGT2GS	Time until pending: 2.321s, Completed successfully without complications
	AC variable displacement valve		Not found	Not found	-	-	-	-	N/A	N/A	N/A	ARACVDGS	, Pending was not set on CAN-bus, Pending was not set in INCA

Bilagor

Tabell 5, Exempelrapport – den första felkoden har inte hittats i felkodsminnet, den tredje felkodens status har inte ändrats till *Confirmed,* den fjärde och femte felkoden är inte emissionsrelaterad därmed är inte MIL relevant så ”MIL OK?”, ”MIL turned off?” och ”Fault code healed?” blir gul markerade

Fault code	Component	UDS_DTC	Expected found? (INCA)	Expected found? (CAN)	Pending? (INCA)	Pending? (CAN)	Confirmed? (INCA)	Confirmed? (CAN)	MIL OK? (INCA)	MIL turned off?	Fault code healed?	Fault codes found	Notes
CACPR	Ambient temperature sensor -	061B02	Not found	Not found	-	-	-	-	N/A	N/A	N/A	STMTR	, Pending was not set on CAN-bus, Pending was not set in INCA
CAEELEA	Camshaft Sensor Exhaust	036500	OK	OK	OK	OK	OK	OK	OK	OK	OK	STMTR, CAEELEA	Time until pending: 1.946s, Completed successfully without complications
ARTCVOP	Vacuum Regulator Turbo Wastegate	004500, 061B02	OK	OK	OK	OK	Became confirmed in cycle: None	Became confirmed in cycle: None	N/A	N/A	N/A	STMTR, ARTCVOP	Time until pending: 2.302s, Confirmed was not set on CAN-bus, Confirmed was not set in INCA
ARTCVGS	Vacuum Regulator Turbo Wastegate	004700, 061B02	OK	OK	OK	OK	OK	OK	N/A	N/A	N/A	STMTR, ARTCVGS	Time until pending: 2.331s, Completed successfully without complications
ARTCVBS	Vacuum Regulator Turbo Wastegate	004800, 061B02	OK	OK	OK	OK	OK	OK	N/A	N/A	N/A	STMTR, ARTCVBS	Time until pending: 2.767s, Completed successfully without complications

Tabell 6, Exempelrapport – den första felkoden har inte hittats, den tredje felkodens status har inte ändrats till *Confirmed*, den fjärde och den femte felkoden har fungerat som de ska då de inte är emissionsrelaterade

Fault code	Component	UDS_DTC	Expected found? (INCA)	Expected found? (CAN)	Pending? (INCA)	Pending? (CAN)	Confirmed? (INCA)	Confirmed? (CAN)	MIL OK? (INCA)	MIL turned off?	Fault code healed?	Fault codes found	Notes
CACPR	Ambient temperature sensor -		Not found	Not found	-	-	-	-	N/A	N/A	N/A		, Pending was not set on CAN-bus, Pending was not set in INCA
CAEELEA	Camshaft Sensor Exhaust	036500	OK	OK	OK	OK	OK	OK	OK	OK	OK	CAEELEA	Time until pending: 2.186s, Completed successfully without complications
ARTCVOP	Vacuum Regulator Turbo Wastegate	004500	OK	OK	OK	OK	Became confirmed in cycle: None	Became confirmed in cycle: None	N/A	N/A	N/A	ARTCVOP	Time until pending: 2.334s, Confirmed was not set on CAN-bus, Confirmed was not set in INCA
ARTCVGS	Vacuum Regulator Turbo Wastegate	004700	OK	OK	OK	OK	OK	OK	N/A	N/A	N/A	ARTCVGS	Time until pending: 2.317s, Completed successfully without complications
ARTCVBS	Vacuum Regulator Turbo Wastegate	004800	OK	OK	OK	OK	OK	OK	N/A	N/A	N/A	ARTCVBS	Time until pending: 2.686s, Completed successfully without complications

Tabell 7, Exempel rapport – CAN-bussen har inte varit funktionell för någon felkod, den första felkoden har inte heller hittats i felkodsminnet med INCA

Fault code	Component	UDS_DTC	Expected found? (INCA)	Expected found? (CAN)	Pending? (INCA)	Pending? (CAN)	Confirmed? (INCA)	Confirmed? (CAN)	MIL OK? (INCA)	MIL turned off?	Fault code healed?	Fault codes found	Notes
CACPR	Ambient temperature sensor -		Not found	Not found	-	-	-	-	N/A	N/A	N/A		, Error when searching in database, Pending was not set on CAN-bus, Pending was not set in INCA
CAEELEA	Camshaft Sensor Exhaust		OK	Not found	OK	-	OK	-	OK	OK	OK	CAEELEA	Time until pending: 2.47s, Error when searching in database
ARTCVOP	Vacuum Regulator Turbo Wastegate		OK	Not found	OK	-	Became confirmed in cycle: None	-	N/A	N/A	N/A	ARTCVOP	Time until pending: 2.313s, Error when searching in database, Pending was not set on CAN-bus, Confirmed was not set in INCA
ARTCVGS	Vacuum Regulator Turbo Wastegate		OK	Not found	OK	-	OK	-	N/A	N/A	N/A	ARTCVGS	Time until pending: 2.344s, Error when searching in database, Pending was not set on CAN-bus
ARTCVBS	Vacuum Regulator Turbo Wastegate		OK	Not found	OK	-	OK	-	N/A	N/A	N/A	ARTCVBS	Time until pending: 2.774s, Error when searching in database, Pending was not set on CAN-bus

Tabell 8, Exempel rapport – test där kommunikationen med INCA inte har fungerat, därmed går det inte att radera felkodsminnet mellan fel injiceringarna det går inte heller att blockera några felkoder

Fault code	Component	UDS_DTC	Expected found? (INCA)	Expected found? (CAN)	Pending ? (INCA)	Pending ? (CAN)	Confirmed? (INCA)	Confirmed? (CAN)	MIL OK? (INCA)	MIL turned off?	Fault code healed?	Fault codes found	Notes
CACPR	Ambient temperature sensor -	004700, 004500, 036500, 004800	-	Not found	-	-	-	-	N/A	N/A	N/A	004700, 004500, 036500, 004800	, Could not connect to INCA, Pending was not set on CAN-bus
CAEELEA	Camshaft Sensor Exhaust	004700, 004500, 036500, 004800	-	OK	-	OK	-	Became confirmed in cycle: 1	N/A	N/A	N/A	004700, 004500, 036500, 004800	, Could not connect to INCA, Confirmed was not set on CAN-bus
ARTCVOP	Vacuum Regulator Turbo Wastegate	004700, 004500, 036500, 004800	-	OK	-	OK	-	Became confirmed in cycle: None	N/A	N/A	N/A	004700, 004500, 036500, 004800	, Could not connect to INCA, Confirmed was not set on CAN-bus
ARTCVGS	Vacuum Regulator Turbo Wastegate	004700, 004500, 036500, 004800	-	OK	-	OK	-	Became confirmed in cycle: 1	N/A	N/A	N/A	004700, 004500, 036500, 004800	, Could not connect to INCA, Confirmed was not set on CAN-bus
ARTCVBS	Vacuum Regulator Turbo Wastegate	004700, 004500, 004800, 036500	-	OK	-	OK	-	Became confirmed in cycle: 1	N/A	N/A	N/A	004700, 004500, 004800, 036500	, Could not connect to INCA, Confirmed was not set on CAN-bus