Tampereen yliopisto

Rasmus Mikkonen

ULKOISEN KIRJASTON INTEGROINTI OSAKSI SIMULINK-SIMULAATIOTA

Moottorinohjaimen software-in-the-loop –simulaattorin integrointi ohjaamaan mallinnettua dieselmoottoria

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Teijo Juntunen
Syyskuu 2024

TIIVISTELMÄ

Rasmus Mikkonen: Ulkoisen kirjaston integrointi osaksi Simulink-simulaatiota
Kandidaatintyö
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Syyskuu 2024

Erilaiset simulointityökalut ovat merkittävä osa nykyaikaista moottorinohjaimen tuotekehitysprosessia. Simulaation avulla voidaan testata ja kehittää moottorinohjaimen toiminnallisuuksia ilman oikeaa moottoria. Näin prosessissa säästetään aikaa ja rahaa sekä parannetaan turvallisuutta, sillä simulaation avulla moottori voidaan asettaa toimintapisteisiin, jotka reaalimaailmassa eivät olisi turvallisia tai edes mahdollisia.

Tässä työssä toteutettiin rajapinta ulkoisen kirjaston ja Matlabin välille sekä integroitiin ulkoisen kirjaston sisältämä moottorinohjaimen software-in-the-loop –simulaattori ohjaamaan olemassa olevaa dynaamista dieselmoottorimallia Simulinkissä. Rajapinnan avulla moottorinohjaimen parametrien sekä signaalien arvoja voidaan lukea sekä uudelleen kirjoittaa Matlabiin luotujen funktioiden avulla. Työssä tutkittiin erilaisia keinoja integroida C-kielellä toteutettuja ohjelmia ja funktioita osaksi Simulink-simulaatiota. Lisäksi esiteltiin software-in-the-loop –simulaation hyötyjä ja haittoja verrattuna model-in-the-loop –simulaatioon sekä esiteltiin sen hyötyjä osana tuotekehitysprosessia.

Toteutetun rajapinnan ja moottorimalli-integraation tuloksena ulkoisen kirjaston sisältämää moottorinohjaimen software-in-the-loop –simulaattoria voidaan simuloida Simulinkissä dynaamisen moottorimallin kanssa, jolloin jo käännettyä moottorinohjaimen ohjelmakoodia voidaan testata mallinnetulla moottorilla ilman hardware-in-the-loop –simulaattoria. Työn tuloksena tuotettiin moottorinohjaimen testauksen ja kehityksen avuksi simulaatioalusta.

Avainsanat: software-in-the-loop, Simulointi, Simulink, Matlab, Moottorinohjain (ECU)

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -ohjelmalla.

ABSTRACT

Rasmus Mikkonen: External Library integration to the Simulink Simulation
Bachelor‘s thesis
Tampere University
Faculty of Engineering and Natural Sciences
September 2024

Various simulation tools are a significant part of the modern engine controller development process. Through simulation, the functionalities of the engine controller can be tested and developed without the need for a real engine. This approach saves time and money and improves safety, as simulation allows the engine to be placed in operating conditions that would be unsafe or even impossible in the real world.

In this work, an interface was implemented between an external library and Matlab, and the engine controller's Software-in-the-loop (SIL) simulator contained in the external library was integrated to control an existing dynamic diesel engine model in Simulink. Using this interface, the values of engine controller parameters and measurement channels can be read and written via functions created in Matlab. The study explored various methods for integrating programs and functions written in the C language into Simulink simulation. Additionally, the benefits and drawbacks of Software-in-the-loop simulation compared to model-in-the-loop (MIL) simulation were presented, along with its advantages as part of the development process.

As a result of the implemented interface and engine model integration, the engine controller's Software-in-the-loop simulator contained in the external library can be simulated in Simulink with the dynamic engine model. This allows the already compiled engine controller software code to be tested against the modeled engine without the need for Hardware-in-the-loop (HIL) simulation. As a result of the work, a simulation platform was produced to aid in the testing and development of the engine controller.

Keywords: Software-in-the-loop, Simulation, Simulink, Matlab, Electronic Control unit (ECU)

The originality of this thesis has been checked using the Turnitin Originality Check service.

ALKUSANAT

Tämä kandidaatintyö tehtiin Agco Power Oy:n Engine Control -osastolle. Haluankin kiittää erityisesti Markus Rantasta mahdollisuudesta kirjoittaa työ mielenkiintoisesta aiheesta sekä Miikka Jormanaista työn aikaisesta avusta. Lisäksi haluaisin kiittää Tampereen yliopiston Teijo Juntusta työni ohjaamisesta.

Tampereella, 13.9.2024

Rasmus Mikkonen

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. TEORIA 3
2.1 Moottorinohjain ECU 3
2.2 Erilaiset simulaattorit 4
2.3 Dynamic Link Library 6
3. TYÖN SUORITUS 8
3.1 Integrointi tavat 8
3.1.1 S-function builder 9
3.1.2 C-Caller 9
3.1.3 Ulkoinen kirjasto 11
3.2 Matlabin ”Lib.pointer” -osoittimet 12
3.3 DLL:n suorittaminen osana Matlab funktio lohkoa 14
3.4 Rajapinnan yksinkertainen testaus Simulinkissä 16
3.5 Simulaattorin integrointi osaksi moottorimallia 20
3.6 SIL-rajapinnan testaus moottorimallin kanssa 23
4. TULOKSET 28
4.1 Vertailu 28
4.2 Hyödyt tuotekehityksessä 28
5. YHTEENVETO 31
LÄHTEET 32

KUVALUETTELO

Kuva 1. Signaalin kulku ja ECU:n yksinkertaistettu rakenne. Muokattu lähteestä [8, s.274] ... 3
Kuva 2. HIL-simulaation periaatekuva. Muokattu lähteestä [1, s. 407] ... 4
Kuva 3. SIL-simuloinnin periaate kuva. Muokattu lähteestä [1, s. 407] ... 5
Kuva 4. MIL-simulaation periaate kuva. Muokattu lähteestä [3, s. 407] ... 5
Kuva 5. C-kääntäjän toiminta yksinkertaistettuna. ... 6
Kuva 6. "S-function builder"- lohko osana yksinkertaista simulaatiota ... 9
Kuva 7. "C Caller" -lohkon sisältö kolmella esimerkkifunktiolla ... 10
Kuva 8. Simulinkin generoima lohko esimerkkifunktiosta ... 10
Kuva 9. Vuokaavio Matlabin ja ulkoisen kirjaston toiminnasta ... 11
Kuva 10. Kaaviokuva Matlab funktiosta rajapinnan yksinkertaiseen testaukseen ... 15
Kuva 11. Moottorinohjauksen päätilakoneen tila ajan funktiona HIL-simulaattorissa ... 16
Kuva 12. Moottorin pyörimisnopeus ajan funktiona HIL-simulaattorissa ... 17
Kuva 13. Sytytysvirran status ajan funktiona HIL-simulaattorissa ... 17
Kuva 14. Moottorinpyörimisnopeus lohkon herätesignaalina ... 18
Kuva 15. Sytytysvirran status lohkon herätesignaalina ... 19
Kuva 16. Rajapinnan kautta haettu vaste moottorinohjauksen päätilakoneen tilasta ... 19
Kuva 17. Periaatekuva toteutettavasta integraatiosta ... 21
Kuva 18. "DLL_ECU Subsystem" -lohko Simulinkissä ... 22
Kuva 19. Testin vuokaavio ... 23
Kuva 20. Moottorin pyörimisnopeuden asetusarvo ja todellinen arvo ajan funktioina ... 24
Kuva 21. Moottorinohjauksen päätilakoneen tila ajan funktiona ... 24
Kuva 22. Asetettu kuorma ajan funktiona ... 25
Kuva 23. V-malli ECU:n kehityksessä [1, s.325] ... 29

LYHENTEET JA MERKINNÄT

CAN	engl. Controller Area Network, CAN-väylä
DLL	engl. Dynamic link library, Windows-ympäristössä käytettävä dynaamisesti linkitetty kirjasto
Dylib	engl. Dynamic Shared Library, MacOs-ympäristössä käytettävä dynaamisesti linkitetty kirjasto
ECU	engl. Engine control unit, Moottorinohjain
HIL	engl. Hardware-in-the-loop, järjestelmän oikeita toimilaitteita sisältävä simulaatiotapa
MIL	engl. Model-in-the-loop, mallipohjainen simulaatiotapa
SIL	engl. Software-in-the-loop, virtuaalisessa ympäristössä toteutettu simulaatiotapa
SO	engl. Shared Object library, Linux-ympäristössä käytettävä dynaamisesti linkitetty kirjasto
Uint8	tietotyyppi 8-bittiselle etumerkittömälle kokonaisluvulle

1. JOHDANTO

SIL (engl. Software-in-the-loop) -simulaatio on varsinkin teollisten ohjelmistojen kehityksessä käytettävä simulaatiotapa, jossa jo käännettyä ohjelmakoodia suoritetaan simulaatiossa tietokoneella oikean ajoympäristön sijaan [1, s.407–408]. Tämän työn tapauksessa dieselmoottorin ohjausohjelmistoa simuloidaan kehittäjän tietokoneella oikean moottorinohjaimen (engl. Engine Control Unit, ECU) sijaan [2, s.18–29].

Tässä työssä toteutetaan C-kielellä toteutetun moottorinohjainohjelmiston SIL-simulaattorin integrointi Matlab-ympäristöön. Matlab on Mathworksin tuottama ohjelmisto erityisesti matriisilaskentaan. Se on käytössä monilla eri tekniikan aloilla ajoneuvojen suorituskyvyn simuloinnista erilaisiin mikrobiologian sovelluksiin [3]. Matlab tarjoaa erilaisia lisäosia eri tekniikan aloille ja sovelluksille, kuten ohjausjärjestelmien suunnitteluun. Matlab-ympäristössä voidaan lisäksi suorittaa erillisiä C-kielellä toteutettuja funktioita. [4, s.1]

Työssä on tarkoitus liittää SIL-simulaattori osaksi jo olemassa olevaa yksinkertaista dieselmoottorin dynamiikkamallia (Liite 1: Moottorimalli Simulinkissä). DLL (engl. Dynamic Link Library) on Windows-ympäristössä käytettävä erillinen kirjasto, joka linkitetään käytettävän ohjelman käyttöön dynaamisesti. DLL-kirjasto ladataan ohjelmaa suorittavan prosessin virtuaaliosoitteistoon, josta ohjelma voi kutsua kirjaston sisältämiä funktioita. [5]. Työ toteutetaan Windows-ympäristössä, jolloin käytettävä SIL-simulaattori on käännetty DLL:ksi.

Tavoitteena on rakentaa toimiva rajapinta DLL:n sisältämän SIL-simulaattorin ja Matlab-ympäristön välille, jolloin simulaattorista tulevia signaalien arvoja voidaan lukea Simulinkissä olevaan moottorimalliin sekä moottorimallin antamia signaalien arvoja voidaan viedä simulaattorille. Simulink on Matlabin lisäosa, joka on tarkoitettu erilaisten järjestelmien simulointiin ja suunnitteluun lohkokaaviopohjaisesti. Simulinkin etuna verrattuna muihin samanlaisiin ohjelmistoihin on tiivis integraatio Matlabiin, jolloin sen ominaisuuksia voidaan hyödyntää Simulinkissä [6, s.341]. Simulaattorin signaalien ja parametrien arvoja, joita moottorimalli ei tarvitse, voidaan lukea ja muokata rajapintaa hyödyntäen. Moottorimalli ei esimerkiksi tarvitse asetettua päästötason parametriarvoa

toimiakseen. Signaalien ja parametrien arvoja voidaan lukea ja kirjoittaa työssä toteutettavilla funktioilla. Moottorimallille tarpeettomat signaali- ja parametriarvot vaikuttavat epäsuorasti signaaleihin, joita moottorimalli tarvitsee, joten mahdollisuus niiden muokkaamiseen ja lukemiseen on moottorinohjausohjelmiston testauksen ja kehittämisen kannalta tärkeää. Työn suoritusosuudessa tutkitaan ja esitellään myös kaksi vaihtoehtoista tapaa integroida C-kielellä toteutettuja ohjelmia Matlab-ympäristöön. Työssä esitettävät vaihtoehtoiset tavat ovat S-function builder ja C-caller [7, 12].

Rajapinnan rakentamisessa tulee huomioida signaalien ja parametrien suuri määrä moottorinohjausohjelmistossa. SIL-simulaattorissa parametreja ja signaaleita on implementoitu noin 14 000. Tässä työssä oletetaan, että SIL-simulaattori on valmiiksi käännetty DLL:ksi, joten sen käännösprosessi rajataan työn ulkopuolelle.

Työn toinen luku sisältää teoriaosuuden, jossa esitellään yleisesti moottorinohjainta sekä erilaisia simulaatiotapoja, joita käytetään hyväksi teollisten ohjaus- ja säätösovellusten tuotekehityksessä. Lisäksi esitellään työhön valittua ulkoista kirjastoa yleisesti. Työn kolmannessa luvussa esitellään kolme erilaista tapaa integroida C-kielellä toteutettuja ohjelmia Simulink-simulaatioon. Lisäksi käsitellään rajapinnan rakentaminen, sen yksinkertainen testaus, SIL-simulaattorin integrointi moottorimallin rinnalle ja sen testaus moottorimallin kanssa. Neljännessä luvussa esitellään saavutettuja hyötyjä ja haittoja MIL-simulaatioon verrattuna sekä käsitellään toteutetun integraation hyötyjä tuotekehityksessä. Työn viimeisenä lukuna on yhteenveto-luku, jossa käsitellään lyhyesti tehty työ sekä esitellään mahdollisia jatkokehitysmahdollisuuksia.

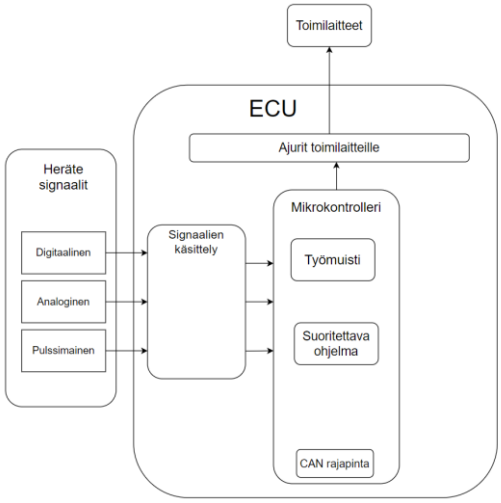
2. TEORIA

Työn toisessa eli teorialuvussa esitellään yleisesti moottorinohjainta sekä erilaisia simulaatiotapoja, joita voidaan käyttää hyväksi tuotekehityksessä. Lisäksi esitellään yleisesti DLL-kirjaston toimintaa.

2.1 Moottorinohjain ECU

Moottorinohjain (eng. Engine Control Unit, ECU) on moottorin elektroniseen ohjaamiseen tarkoitettu kokonaisuus, joka koostuu erilaisista komponenteista ja rajapinnoista. ECU:ssa on sisällytettynä ohjelmisto, joka vastaa moottorinohjauksesta. Tätä ohjausohjelmistoa suoritetaan ECU:n sisällä olevalla mikrokontrollerilla. [8, s.272–277] Ohjelmisto laskee esimerkiksi ohjausarvoja moottorin toimilaitteille ja suorittaa mahdolliset säätöalgoritmit moottorin osajärjestelmille. Ohjausohjelmisto sisältää huomattavan suuren määrän erilaisia parametreja, joilla ohjelmiston konfiguraatiota voidaan muuttaa erilaisia moottoreita varten.

ECU:ssa on toteutettuna myös herätesignaalien käsittely ja tarkistus, jonka tarkoituksena on poistaa häiriöitä signaalista ja muokata signaalin taso mikrokontrollerille sopivaksi [8, s.273]. Signaalin kulku ja ECU:n yksinkertaistettu rakenne on esitetty kuvassa 1, jossa heräte signaalit lohkossa olevat signaalit ovat moottorin erilaisilta antureilta tulevia mittaussignaaleja. Signaalit käsitellään mikrokontrollerille sopiviksi signaalien käsittely-lohkossa, jonka jälkeen suoritettavan ohjelman perusteella mikrokontrolleri laskee ohjausarvoja moottorin toimilaitteille.



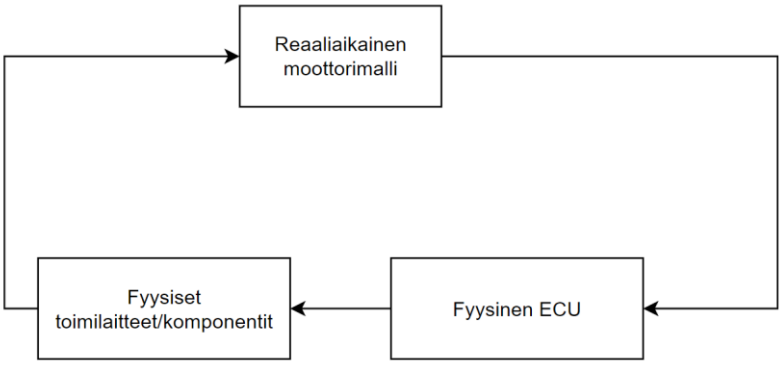
Kuva 1. *Signaalin kulku ja ECU:n yksinkertaistettu rakenne. Muokattu lähteestä [8, s.274]*

Tässä työssä käytetty moottorinohjaimen SIL-simulaattori sisältää ainoastaan ECU:n ohjelmistototeutuksen. Esimerkiksi CAN (engl. Controller Area Network) -kommunikaatiota, ECU:n elektroniikan dynamiikkaa tai rajapintoja muihin ajoneuvon järjestelmiin ei ole mallinnettu, koska työssä toteutettavan simulaation avulla kehitetään ja testataan vain ECU:n ohjelmistoa.

2.2 Erilaiset simulaattorit

ECU:n ohjelmiston kehitystä tukevat erilaiset simulaattorit eri kehitysvaiheissa. Simulaattorien käytöllä voidaan vähentää oikeiden moottorien käyttöä testipenkeissä. Käytön vähentymisellä saadaan kehityksestä kustannustehokkaampaa, turvallisempaa ja ympäristöystävällisempää [9, s. 39]. Alla olevissa kappaleissa esitellään erilaisia simulointitapoja moottorinohjainohjelmiston kehityksen näkökulmasta.

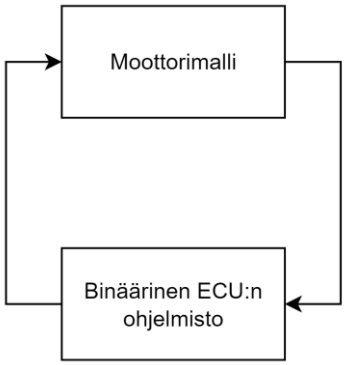
HIL (engl. Hardware-in-the-loop) on simulaattorijärjestelmä, jossa osa simuloitavan järjestelmän komponenteista on fyysisiä ja osa mallinnettu virtuaaliseksi. Fyysisiä komponentteja voivat olla esimerkiksi erilaiset venttiilit. HIL-simulaatiojärjestelmässä on mukana myös oikea fyysinen ECU, joka ohjaa fyysisiä toimilaitteita, samoin kuin oikeassa järjestelmässäkin. HIL-simuloinnissa järjestelmän mallilta vaaditaan reaali-aikaisuutta fyysisten komponenttien takia. [1, s. 407–408; 9, s. 5–6] Havainnekuva HIL-järjestelmästä on esitetty kuvassa 2.



Kuva 2. *HIL-simulaation periaatekuva. Muokattu lähteestä [1, s. 407]*

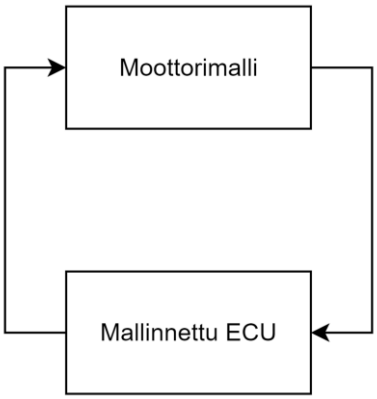
SIL (engl. Software-in-the-loop) on simulaattorijärjestelmä, jossa kaikki järjestelmän komponentit ovat virtuaalisia. Simulaatiossa suoritetaan jo käännettyä ohjelmakoodia

virtualisoidulla moottorinohjaimella. SIL-simulaatiota voidaan simuloida kehittäjän omalla tietokoneella, jolloin säästetään HIL-järjestelmässä käytettävää aikaa. Koska simulaatiossa ei ole mukana oikeita fyysisiä toimilaitteita, reaaliaikavaatimustakaan ei tällöin SIL-simuloinnissa ole. [1, s. 407–408] Simulaatiota voidaan tietyissä tilanteissa ajaa jopa reaali-aikaa nopeammin simulaation toteutustavan mukaan. Tässä työssä käytetyssä SIL-simulaattorissa osa ECU:n ohjelmistosta on käännetty DLL:n sisältämälle simulaatioisännän arkkitehtuurille, kohderaudan eli ECU:n arkkitehtuurin sijaan. Havainnekuva SIL-järjestelmästä on esitetty kuvassa 3.



Kuva 3. *SIL-simuloinnin periaate kuva. Muokattu lähteestä [1, s. 407]*

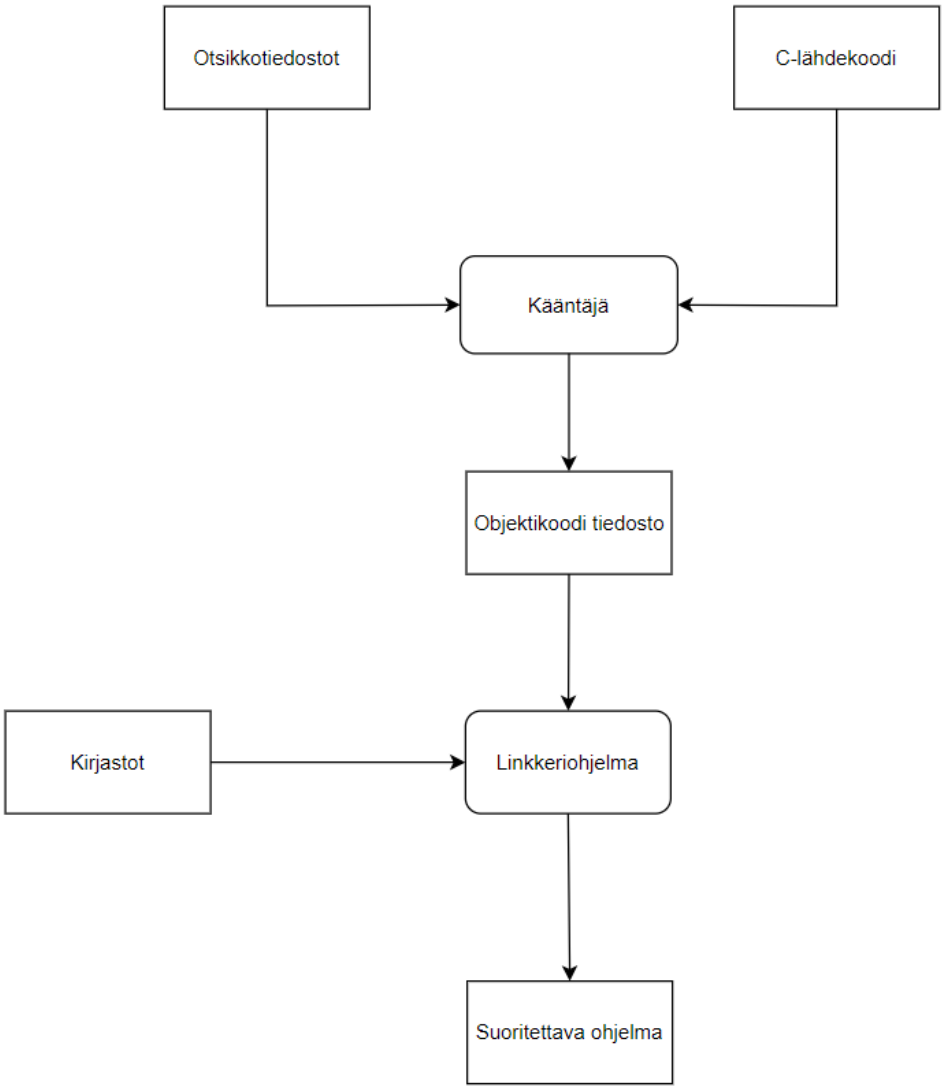
MIL (engl. Model-in-the-loop) -simulaatiossa käytetään moottorinohjaimen kehitysaikaista mallia, jota voidaan simuloida moottorimallin kanssa. Erona SIL-simulaatioon on, että ECU:n ohjelmistoa ei ole käännetty suoritettavaksi ohjelmaksi, vaan mallinnettu ohjelmisto on esimerkiksi Simulink-ympäristössä, jossa sitä voidaan simuloida. [1, s. 407–408] Havainnekuva MIL-järjestelmästä on esitetty kuvassa 4.



Kuva 4. *MIL-simulaation periaate kuva. Muokattu lähteestä [3, s. 407]*

2.3 Dynamic Link Library

Ohjelmoinnissa kirjastot ovat tapa tuoda ohjelmakoodiin ulkopuolisia toiminnallisuuksia. Kirjastoja ei voida suorittaa itsenäisesti, mutta niitä voidaan sisällyttää ohjelmiin toimintojen lisäämiseksi. Ohjelmaan lisätään kirjaston otsikkotiedosto, mutta ohjelman käännösvaiheessa kirjaston lähdekoodia ei käännetä, vaan se tuodaan ohjelmaan jo valmiiksi käännettynä. Kun ohjelma käännetään, kirjastot lisätään kääntäjässä olevan linkkausosan avulla osaksi käännettyä ohjelmakoodia. Kääntäjän toimintaa kirjastojen lisäämisessä on esitetty yksinkertaistettuna kuvassa 5.



Kuva 5. *C-kääntäjän toiminta yksinkertaistettuna.*

Kirjastoja voidaan linkata ohjelmaan staattisesti tai dynaamisesti. Dynaamisesti linkattu kirjasto (DLL) ladataan yleisesti suorittavan laitteen muistiin, jolloin kaikki sitä tarvitsevat sovellukset voivat käyttää kirjaston toimintoja. DLL:n objektikoodia ei kuitenkaan

sisällytetä suoritettavaan ohjelmaan toisin kuin staattisesti linkitetyn kirjaston, jolloin se voidaan korvata päivitetyllä tai korjatulla versiolla kääntämättä itse pääohjelmaa. [5]

3. TYÖN SUORITUS

Työn suoritusvaiheessa esitellään kolme erilaista tapaa integroida C-kielellä toteutettuja ohjelmia Simulink-simulaatioon. Tässä luvussa rakennetaan myös rajapinta sekä tarvittavat luku- ja kirjoitusfunktiot ulkoisen DLL-kirjaston ja Matlabin välille. Lisäksi testataan rajapinnan toimivuus ja integroidaan DLL:n sisältämä SIL-simulaattori ohjaamaan jo olemassa olevaa Agco Powerin kehittämää dieselmoottorin dynaamista mallia Simulinkissä [11, s.23–29].

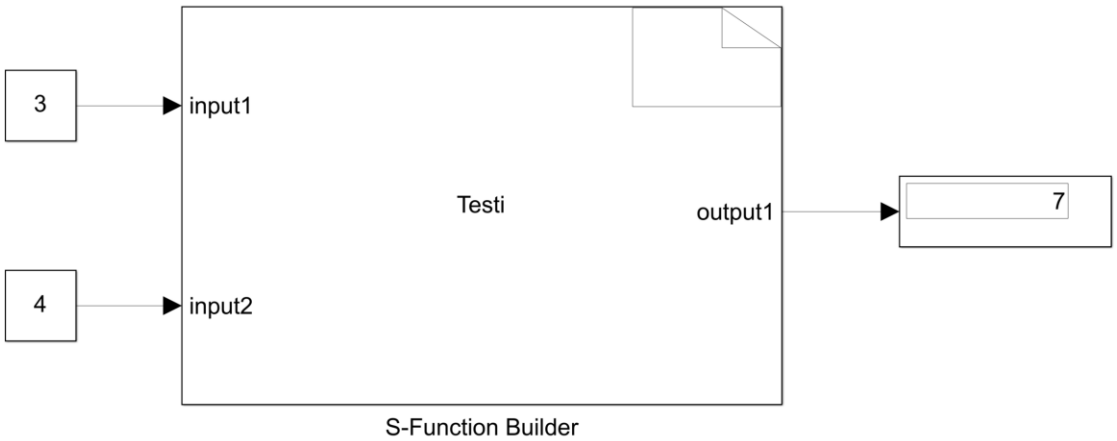
3.1 Integrointi tavat

Erillistä C-kielellä toteutettua koodia tai kokonaisen ohjelman voi tuoda Matlab-ympäristöön erilaisilla tavoilla, jotka esitellään alaluvuissa 3.1.1–3.1.3. Kaikki esitellyt metodit ovat osa maksullista Matlabin perusversiota. Alaluvussa 3.1.3 käytetään hyväksi työssä käytettävää DLL-kirjastoa. Alaluvuissa 3.1.1 ja 3.1.2 käytetään hyväksi yksinkertaista C-kielellä toteutettua ohjelmaa, joka sisältää kolme yksinkertaista aritmeettista funktiota:

```
int testi_summa(int luku1, int luku2)
{
// lisätään luvut toisiinsa
return luku1 + luku2
}

int testi_tulo(int luku1, int luku2)
{
// kerrotaan luvut keskenään
return luku1 * luku2
}

int testi_jako(int luku1, int luku2)
{
// jaetaan ensimmäinen luku toisella
return luku1 / luku2
}
```

3.1.1 S-function builder

S-Function Builder on Simulinkin käyttäjän määrittämät funktiot (engl. User-Definied Functions) -kirjastosta löytyvä lohko, jolla voidaan automaattisesti integroida C-koodia käännettävään S-funktioon. Lohkon toimintaa voidaan muuttaa lohkon asetuksista. [12] Lohkoon voidaan sisällyttää ulkopuolista lähdekoodia kuten normaaliin C-kielellä toteutettuun ohjelmaan. Tässä esimerkissä lohkoon on sisällytetty aiemmin esitetty yksinkertainen joukko aritmeettisia funktioita. Lohkon ulostuloksi tuleva arvo määritellään “S-function builder”-lohkon asetuksissa. Tässä tapauksessa lohkon ainoaan ulostuloporttiin ”output1” määritetään funktion ”testi_summa” paluuarvo parametreilla, jotka saadaan lohkon kahdesta sisääntulosta ”input1” ja ”input2”.
Toimiakseen simulaatiossa lohkon sisäänmeno ja ulostuloporttien sekä niihin liitettyjen Simulink-elementtien tietotyypit tulee määrittää yhdenmukaisiksi. Kuvassa 6 on esitetty ”S-function builder” -lohko osana yksinkertaista simulaatiota, jossa lohkon sisäänmenoihin on määritetty kaksi vakioarvoista muuttujaa, lohkon ulostulo esitetään Display-lohkossa.

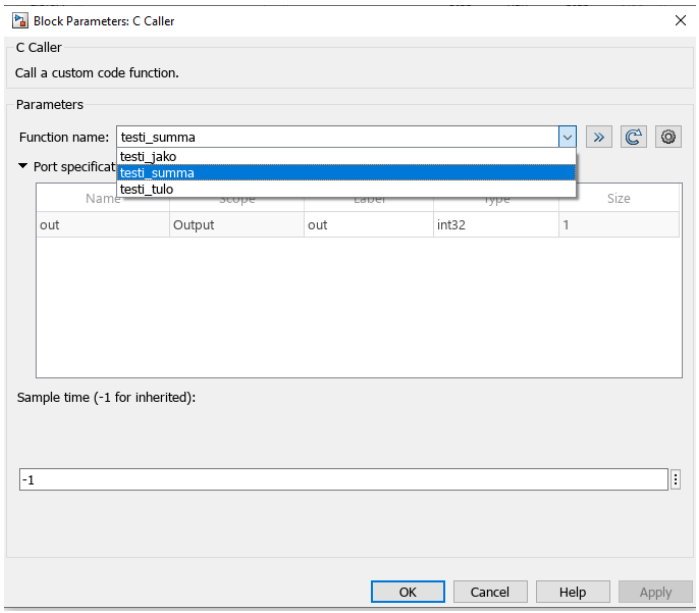


Kuva 6. *“S-function builder”- lohko osana yksinkertaista simulaatiota*

3.1.2 C-Caller

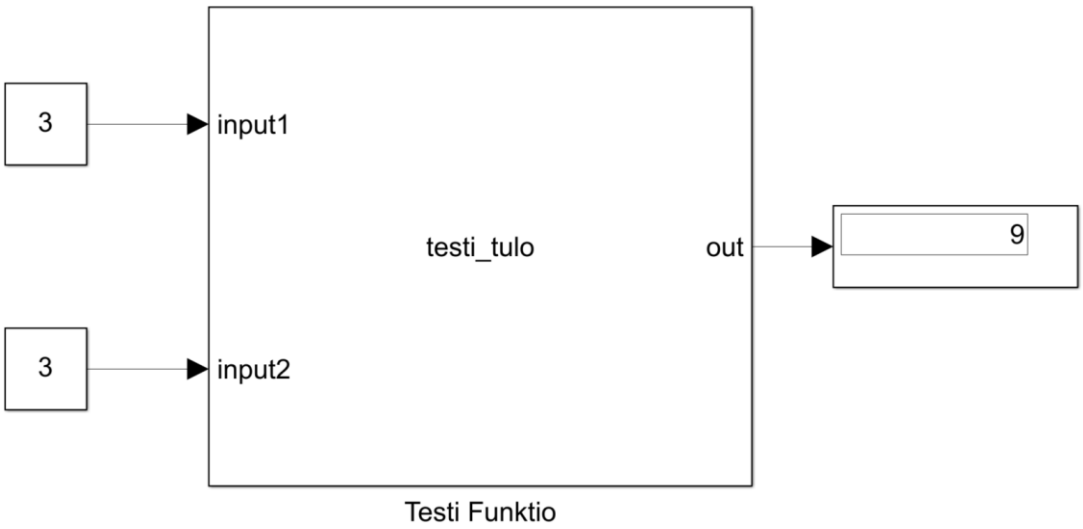
C Caller on simulinkin “User-Defined Functions” -kirjastosta löytyvä lohko, jolla voidaan kutsua erillisestä lähdekoodista löytyvää funktiota. “C Caller”- lohkon asetuksiin määritellään sisällytettävä otsikko- ja lähdekooditiedosto sekä tarvittaessa niiden hakemistopolut. Lohkoon määritellään pudotusvalikosta kutsuttava funktio, jonka Simulink parsii automaattisesti, jos otsikko- ja lähdekooditiedostot ovat validit. [12]

Kuvassa 7 on esitetty lohkon sisältö kolmella esimerkkifunktiolla, kun lähdekooditiedostot on määritelty lohkon asetuksiin.



Kuva 7. ”C Caller” -lohkon sisältö kolmella esimerkkifunktiolla

Lähdekoodissa olevan funktion argumentit ja paluuarvot generoituvat Simulink-lohkoon sisään- ja ulostuloiksi. Lähdekoodissa toteutetun funktion ja simulaatiossa olevien elementtien tietotyypit täytyy määrittää yhdenmukaisiksi simulaation toimimisen varmistamiseksi. [12] Kuvassa 8 on esitettynä Simulinkin generoima lohko funktiosta ”testi_tulo”, jolle on annettu kaksi vakioarvoista sisäänmenoa. Simulaation jälkeen lohkon vaste esitetään ”Display” -lohkossa.



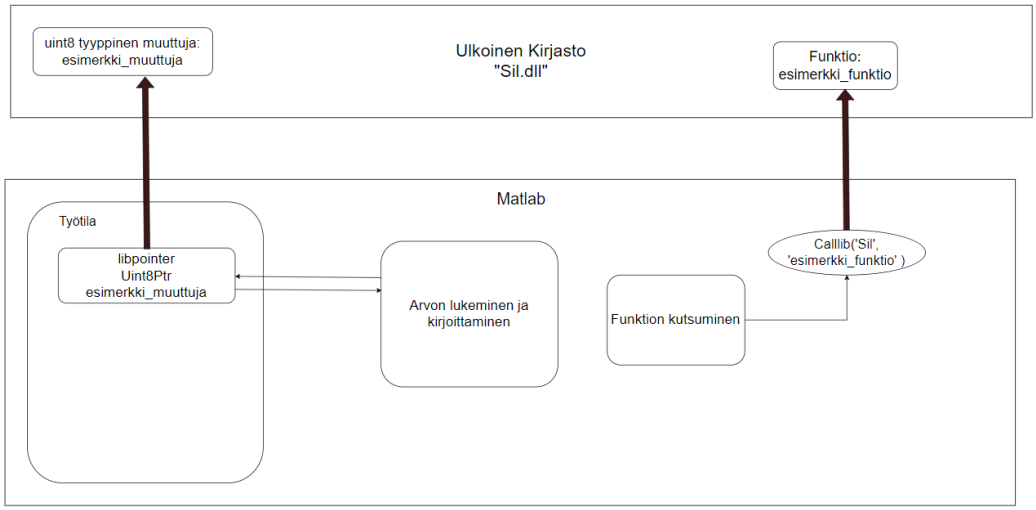
Kuva 8. Simulinkin generoima lohko esimerkkifunktiosta

3.1.3 Ulkoinen kirjasto

Matlab-ympäristöön voidaan ladata ulkopuolinen C-kielinen kirjasto, jonka toiminnallisuuksia, funktioita ja muuttujia voidaan käyttää hyväksi ajonaikaisesti. Windows-ympäristössä Matlab tukee DLL:ää, kun Linux-ympäristössä käytössä on SO-kirjasto (engl. Shared Object Library). Applen macOS käyttöjärjestelmällä Matlab tukee dylib-kirjastoa (engl. Dynamic Shared Library). Työ toteutetaan Windows-ympäristössä, joten työssä on käytetty DLL:ää.

Ulkoinen kirjasto voidaan ladata Matlabin muistiin ”loadlibrary” -funktiolla, jolle annetaan parametreiksi kirjasto- ja otsikkotiedosto. Käytön jälkeen kirjasto poistetaan muistista funktiolla ”unloadlibrary”. Kirjastossa olevat funktiot ja muuttujat saadaan tulostettua komentoikkunaan tai tallennettua muuttujaan funktiolla ”libfunctions”, jolle annetaan parametrina kirjaston nimi.

Kirjaston sisältämiä funktioita voidaan kutsua ”calllib” -funktiolla. Funktiolla annettavat parametrit ovat kirjaston ja funktion nimi sekä funktiolle annettavat kutsuparametrit. ”Calllib” -funktio palauttaa Matlabiin funktion mahdolliset paluuarvot. Kirjastossa olevien muuttujien arvoja voidaan lukea ja muuttaa käyttäen hyväksi Matlabin ”lib.pointer” -osoittimia. Kutsuttaessa kirjaston muuttujaa ”calllib” -funktiolla ja tallentamalla paluuarvo muuttujaan Matlab luo muuttujasta ”lib.pointer” -osoittimen, jolle täytyy määrittää yhteensopiva tietotyyppi kirjastossa olevan muuttujan kanssa. Esimerkiksi ”uint8” -tyyppisen muuttujan ”lib.pointer” -osoittimen tietotyypiksi määritetään ”uint8Ptr”. Kirjastossa olevan funktion kutsumista ja muuttujan arvon lukemista on esitetty vuokaaviona kuvassa 9. [12]



Kuva 9. Vuokaavio Matlabin ja ulkoisen kirjaston toiminnasta

3.2 Matlabin ”Lib.pointer” -osoittimet

Matlab kommunikoi ulkoisessa kirjastossa olevien globaalien muuttujien kanssa käyttäen hyväkseen ”lib.pointer” -osoittimia. Matlabin työtilaan luodaan ”lib.pointer” -osoitin jokaista ulkoisen kirjaston globaalia muuttujaa varten, jotka osoittavat haluttuihin muuttujiin kirjastossa. Niiden avulla pystytään lukemaan ja kirjoittamaan muuttujien arvoja kirjastosta luvun 3.1.3 mukaisesti.

”Lib.pointer” -osoitin luodaan Matlabin työtilaan libpointer-funktiota käyttäen [12]. Tässä työssä käytettävässä DLL:ssä globaalien muuttujien määrä on yli 14000, jolloin osoittimien luonti ja alustus tulee suorittaa ohjelmallisesti. DLL sisältää erityyppisiä muuttujia dimensioiden sekä tietotyyppien osalta, jotka tulee ottaa huomioon osoittimia alustaessa. Tässä työssä toteutetaan mahdollisuus muokata ja lukea rajapinnan kautta vain skalaarimuuttujia, jolloin suuremman dimension omaavien muuttujien lukeminen ja muokkaaminen on rajattu työn ulkopuolelle.

Luvussa 3.1.3 esitetään kuinka DLL:n sisältämät funktiot ja muuttujat voidaan tallentaa Matlabin tietorakenteeseen käyttämällä ”libfunctions” -funktiota. Työssä käytetty ohjelmakoodi ”lib.pointer” -osoittimien luomiseen on esitetty algoritmissa 1.

```
1    // tallennetaan ulkoisen kirjaston funktiot ja muuttujat
     muuttujaan ”functions”
2    functions = libfunctions(string kirjaston nimi)

3    // poistetaan etuliite ”lib.pointer” kaikista muuttujan
     //”functions” soluista ja tallennetaan
4    // muutetut solut muuttujaan ”withoutDatatypes”
5    withoutDatatypes = removeSuffix(”libpointer”, functions)

6    // poistetaan tyhjät solut muuttujasta “withoutDatatypes”
7    withoutDatatypes = removeEmptyCells(withoutDatatypes)

8    // Luodaan jokaisesta soulusta ”lib.pointer” -osoitin ja
     //asetetaan sille tietotyyppi sekä dimensio 1
9    for i in withouDatatypes{
10       withoutDatatypes{i}=makeLibPointer(withouDatatypes{i})
11       setDataType(withDatatypes{i},withDatatype.datatype,1)}
```

***Algoritmi 1**: Lib.pointer osoittimien luonti*

Algoritmissa 1 esitetyssä ohjelmakoodissa tallennetaan ulkoisen kirjaston muuttujan ja funktiot muuttujaan ”functions”, jonka soluilla on etuliite ”lib.pointer”. Algoritmin seuraavassa vaiheessa soluista poistetaan edellä mainittu etuliite ja muutetut solut tallennetaan muuttujaan ”withoutDatatypes”, minkä jälkeen tyhjät solut poistetaan. Viimeisessä vaiheessa jokaisesta solusta luodaan ”lib.pointer” -osoitin omalle nimellä ja tietotyypillään sekä asetetaan dimensioksi arvo 1.

DLL:n sisältämien muuttujien lukemista ja muokkaamista varten luodaan funktiot molemmille toiminnoille. Funktioita tullaan kutsumaan Simulinkin “Matlab Function” -lohkon kautta, joten toteutuksessa tulee ottaa huomioon, että funktioilla on pääsy Matlabin työtilaan. Pääsy voidaan toteuttaa ”evalin” -funktiolla, jolla voidaan muokata ja lukea työtilassa olevia muuttujia erillisiä Matlab-funktioita käyttäen [12].

Muuttujien arvojen lukemiseen luodaan funktio nimeltään “sil_read_value”, joka parametrinaan tarvitsee luettavan muuttujan nimen. Alla on esitetty luku funktion toteutus pseudokoodina ja lopullinen Matlab toteutus liitteessä 2.

```
sil_read_value(string muuttujanNimi)
{
     // lisätään muuttujan nimen perään pääte .value

     muuttujanNimi = muuttujanNimi+'.value'

     // funktio palauttaa työtilassa olevan ”lib.pointer” -
     // osoittimen päässä olevan arvon
     return evalin(workspace, muuttujanNimi)
}
```

Muuttujien arvojen muokkaamiseen luodaan ”sil_set_value” -funktio, joka parametrinaan tarvitsee halutun muuttujan nimen sekä asetettavan arvon. Alla on esitetty muokkaus funktion toteutus pseudokoodina ja lopullinen Matlab toteutus liitteessä 2.

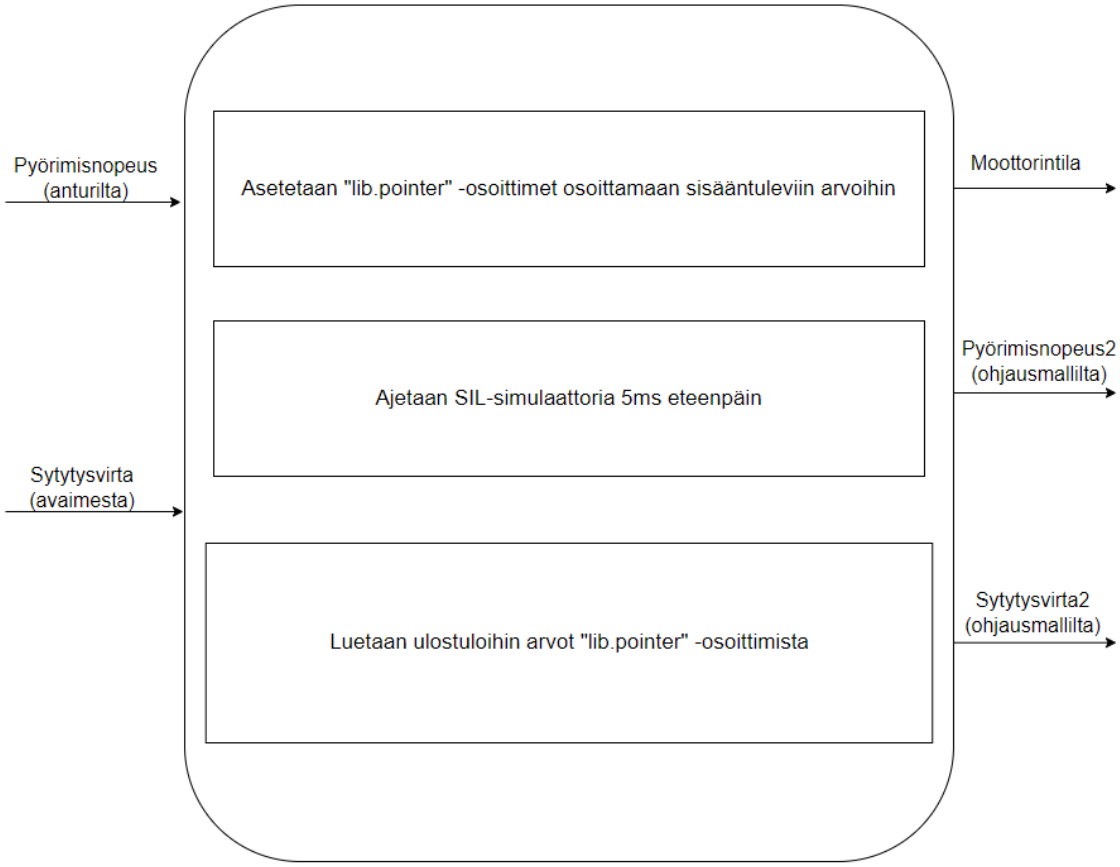
```
sil_set_value(string muuttujanNimi, int asetettava arvo)
{
     // lisätään muuttujan nimen perään pääte .value =
     asetettava arvo

     muuttujan nimi = muuttujan nimi + ”.value” + asetettava
arvo

     // ”evalin” -funktiolla voidaan asettaa haluttu
     // ”lib.pointer” -osoittin osoittamaan asetettavaan
     arvoon.

     evalin(workspace, muuttujan nimi)
}
```

3.3 DLL:n suorittaminen osana Matlab funktio lohkoa

”Matlab Funktio” -lohko on Simulinkin “User-Defined Functions” -kirjastosta löytyvä lohko, jonka avulla käyttäjä pystyy lisäämään Matlab-ohjelmakoodia simulaatioon. Lohkoon määritellään funktio, jonka parametrit generoituvat automaattisesti lohkon sisäänmenoiksi ja funktion paluuarvot lohkon ulostuloiksi. [12]

SIL-simulaattorin hallintaa varten Simulinkiin toteutetaan ”Matlab funktio” -lohko, jonka sisäänmeno signaaleiksi asetetaan kirjoitettavien parametrien ja signaalien arvot. Ulostulo signaaleiksi asetetaan luettavien parametrien ja signaalien arvot.

DLL:n sisältämä SIL-simulaattori sisältää funktion, jolla simulaattoria voidaan ajaa eteenpäin tietty aika millisekunteina, joka annetaan funktiolle kutsuttaessa parametrina. Tässä työssä jokaisella Simulinkin simulaatio askeleella SIL-simulaattoria ajetaan 5 millisekuntia eteenpäin. Rajapinnan yksinkertaista testausta varten lisätään ”Matlab Funktio” -lohko ensin tyhjään Simulink-malliin. Yksinkertainen ”Matlab funktio” -lohko kahdella sisäänmenolla ja kolmella ulostulolla on esitetty kuvassa 10.



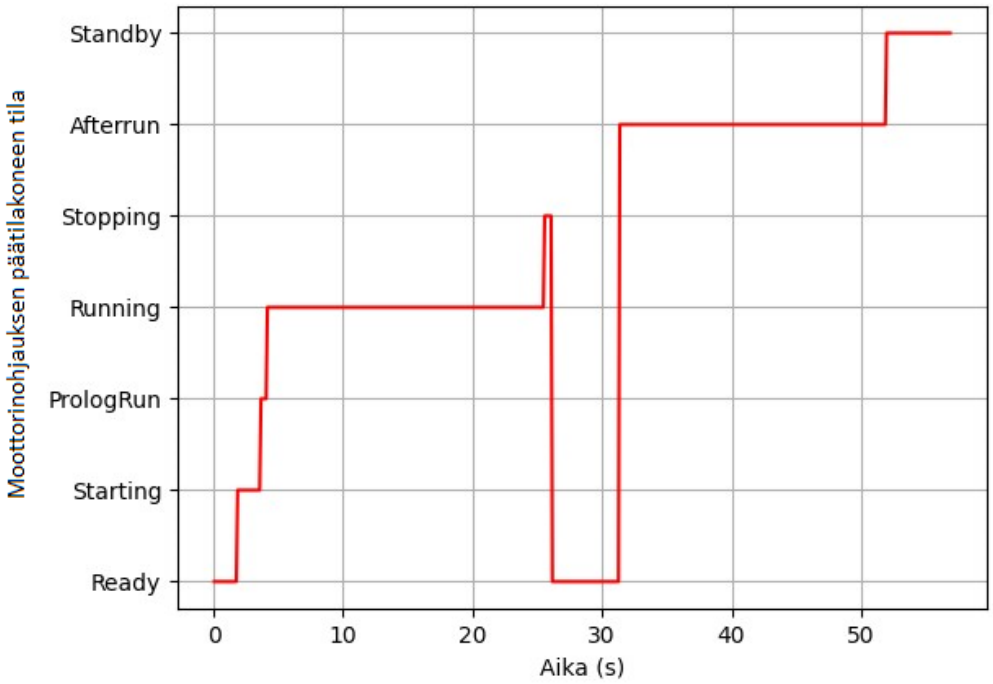
Kuva 10. *Kaaviokuva Matlab funktiosta rajapinnan yksinkertaiseen testaukseen*

Kuvassa 10 esitetyssä lohkossa herätesignaaleina ovat moottorin pyörimisnopeuden arvo luettuna anturilta ja tieto siitä onko sytytysvirta kytketty. Ulostuloina ovat moottorin tämänhetkisen tilan sekä sytytysvirran tilan ja moottorin pyörimisnopeuden kertovat signaalit. Pyörimisnopeuden ja sytytysvirran ulostulo signaalit ovat ECU:n ohjelmiston

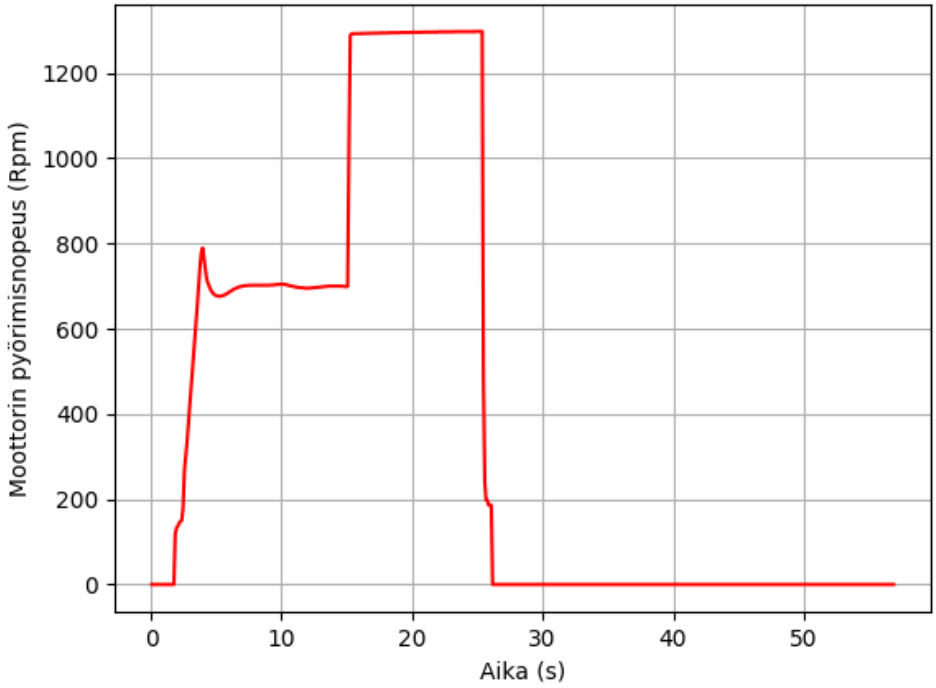
ohjausmallien signaaleja, toisin kuin herätesignaaleina olevat anturiarvot, jotta rajapinnan toimivuudesta voidaan varmistua. Funktiossa asetetaan sisääntulevien signaalien arvot niitä vastaaviin ”lib.pointer” -osoittimiin, minkä jälkeen SIL-simulaattoria ajetaan 5 ms eteenpäin. Lopuksi luetaan ulostuloina olevien signaalien arvot niitä vastaavista ”lib.pointer” -osoittimista.

3.4 Rajapinnan yksinkertainen testaus Simulinkissä

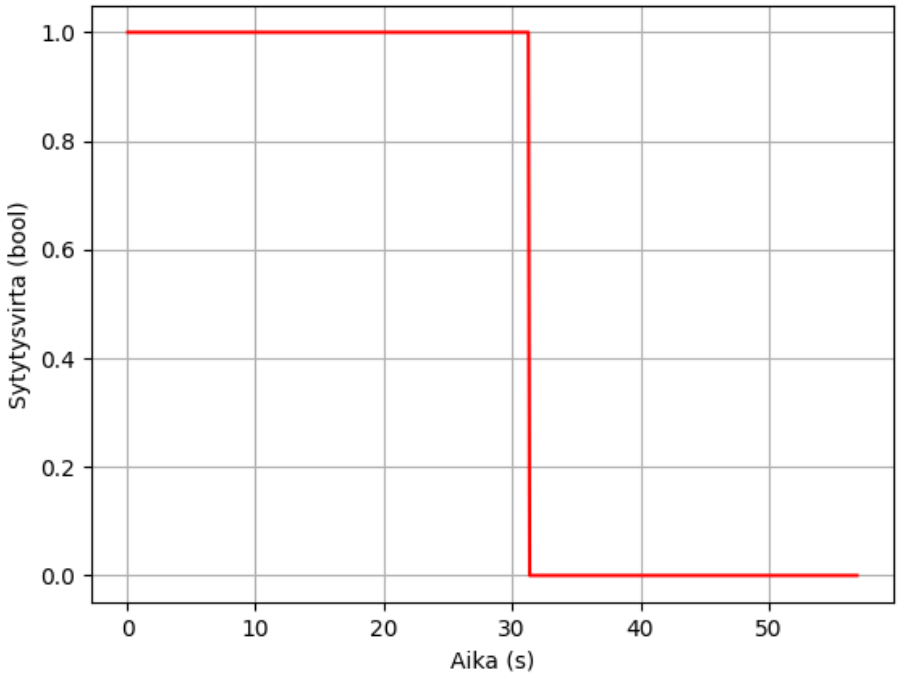
Rajapinnan yksinkertaisessa testauksessa luodaan tilanne, jossa SIL-simulaattorin tulee toimia oikean moottorinohjaimen logiikan mukaisesti. Sen jälkeen tarkastetaan, että simulaatiotulokset vastaavat Agco Powerin moottorinohjaimen logiikkaa. HIL-simulaattorilla nauhoitettiin ECU:n logiikkaa moottorinohjauksen päätilakoneen tilojen osalta moottorin eri toimintapisteissä. Päätilakone määrittää moottorinohjauksen hetkellisen tilan ohjelmiston muille osille. SIL-simulaattorin logiikkaa voidaan verrata HIL-simulaattorilla nauhoitettuun kuvaajaan moottorinohjauksen päätilakoneen tilasta eri toimintapisteissä, joka on esitetty kuvassa 11. HIL-nauhoituksessa käytetty moottorin pyörimisnopeus on esitetty ajan funktiona kuvassa 12 ja sytytysvirran status ajan funktiona kuvassa 13.



Kuva 11. *Moottorinohjauksen päätilakoneen tila ajan funktiona HIL-simulaattorissa*

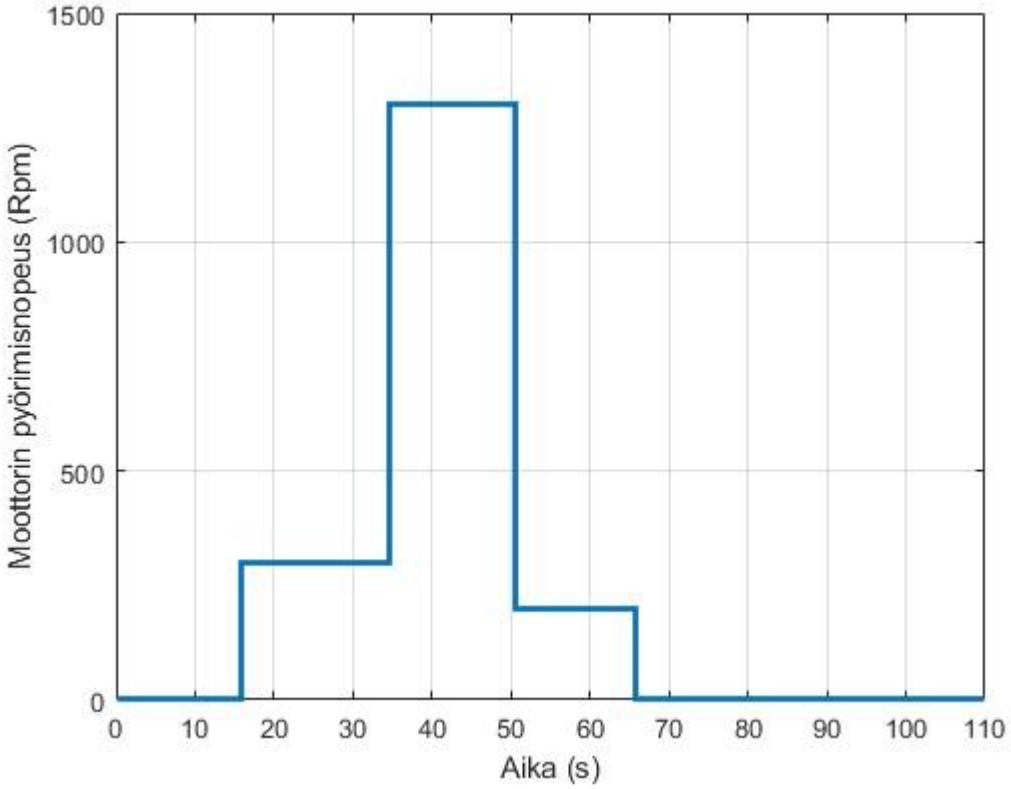


Kuva 12. *Moottorin pyörimisnopeus ajan funktiona HIL-simulaattorissa*

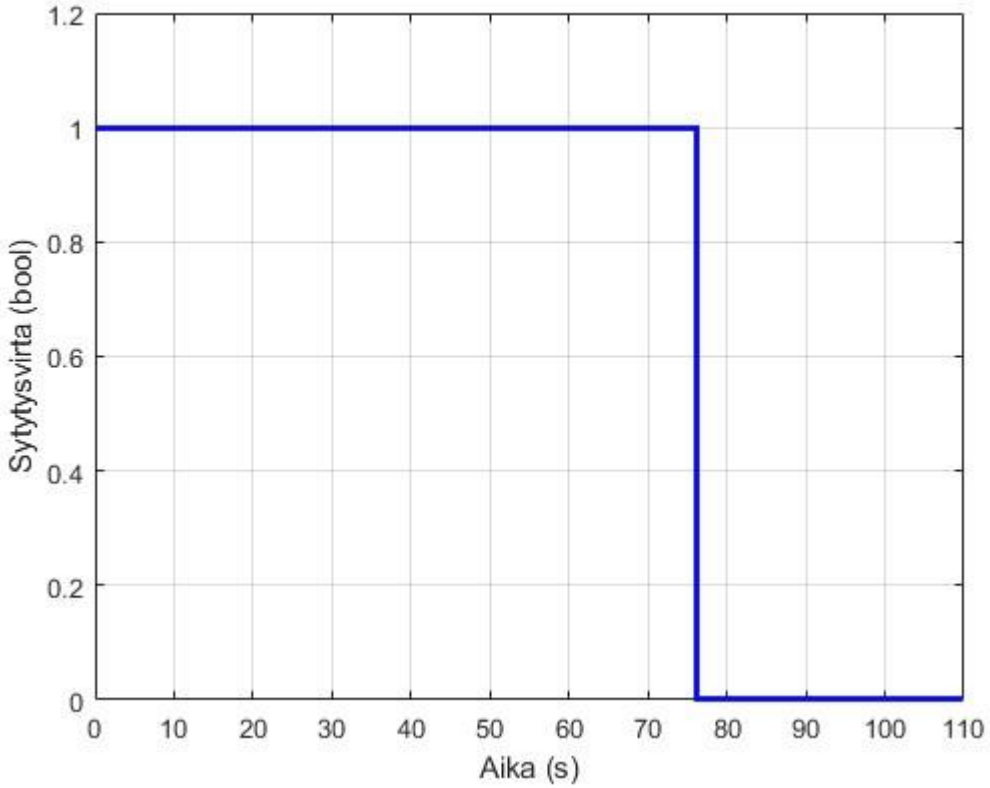


Kuva 13. *Sytytysvirran status ajan funktiona HIL-simulaattorissa*

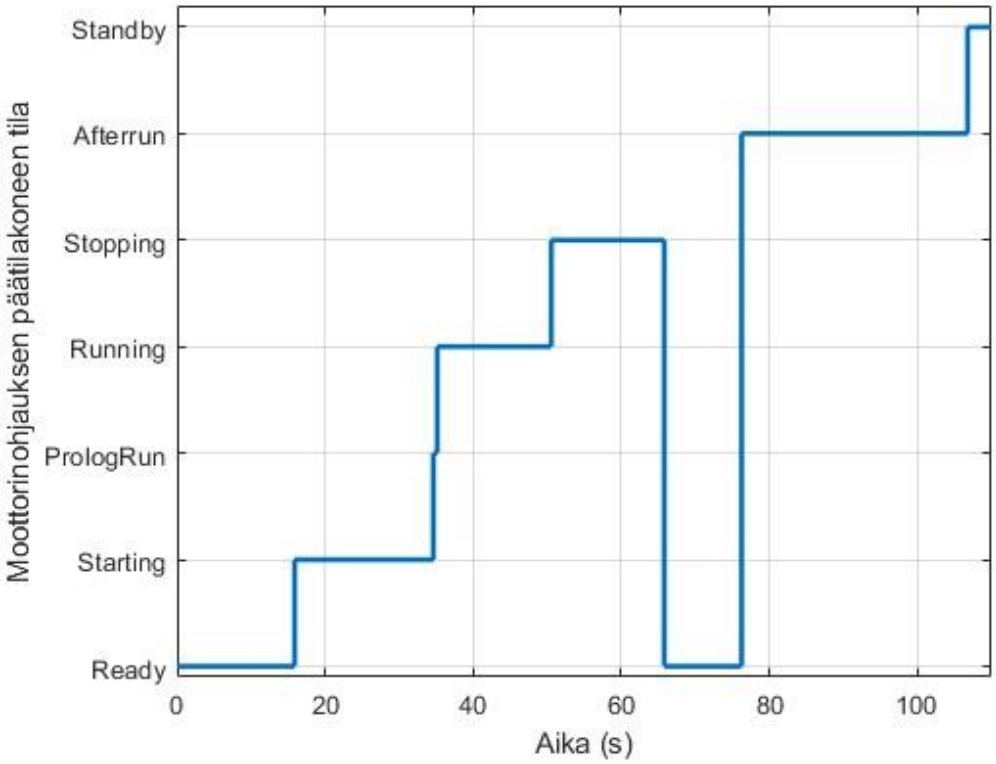
Kuvassa 10 esitettyyn ”Matlab funktio” -lohkoon syötetään herätesignaalina moottorin pyörimisnopeus ja tieto sytytysvirran päällä olosta. Kuvassa 14 on esitetty lohkon herätesignaalina oleva moottorin pyörimisnopeus ajan funktiona. Kuvassa 15 on esitetty lohkon herätesignaalina oleva sytytysvirran tila ajan funktiona. Kuvassa 16 esitetään lohkon vastesignaalina oleva päätilakoneen tila ajan funktiona. Ajanhetket, tapahtumat sekä alku- ja lopputilat ovat taulukoituna taulukossa 1.



Kuva 14. *Moottorinpyörimisnopeus lohkon herätesignaalina*



Kuva 15. *Sytytysvirran status lohkon herätesignaalina*



Kuva 16. *Rajapinnan kautta haettu vaste moottorinohjauksen päätilakoneen tilasta*

Taulukko 1. *Taulukoidut ajanhetket, tapahtumat ja tilat*

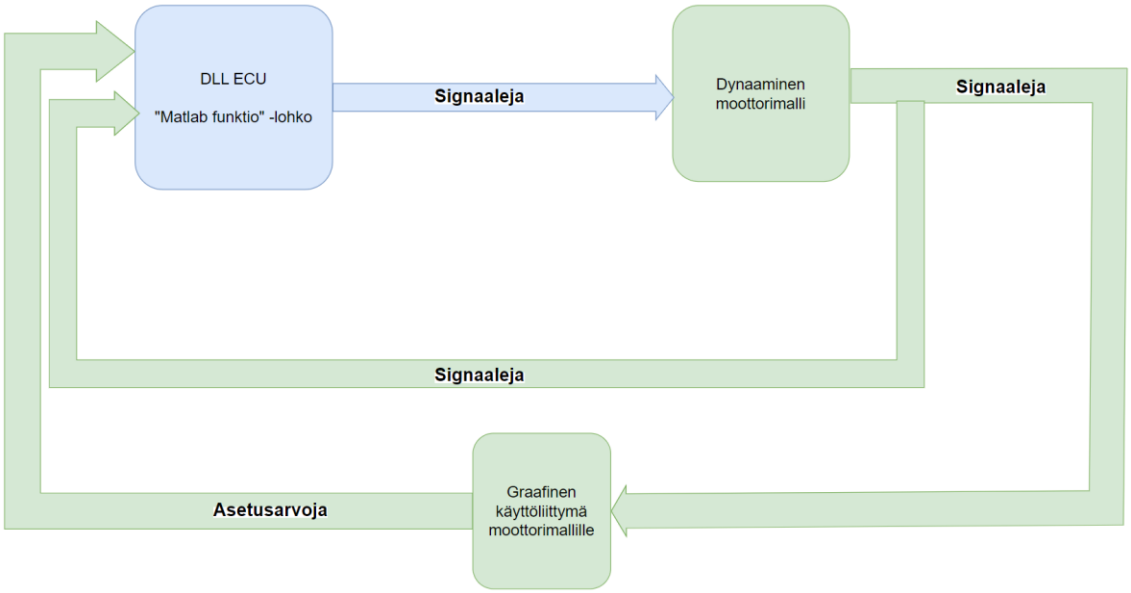
Aika (s)	Alkutila	Lopputila	Tapahtuma
16	Ready	Starting	Pyörimisnopeus nostetaan 300 rpm:ään
35	Starting	PrologRun	Pyörimisnopeus nostetaan 1300 rpm:ään
36	PrologRun	Running	Pyörimisnopeus nostetaan 1300 rpm:ään
50	Running	Stopping	Pyörimisnopeus lasketaan 200 rpm:ään
66	Stopping	Ready	Pyörimisnopeus lasketaan 0 rpm:ään
76	Ready	Afterrun	Sytytysvirta kytketään pois päältä
106	Afterrun	Standby	30 s kulunut ajanhetkestä 76 s

Tila pysyy arvossa *Ready*, kunnes pyörimisnopeus nostetaan arvoon 300 rpm, tällöin tila muuttuu arvoon *Starting* kuten taulukossa 1 on esitetty. Pyörimisnopeus nostetaan arvoon 1300 rpm, jolloin tila muuttuu tilaan *Running* tilan *PrologRun* kautta kuvassa 11 esitetyn toimintalogiikan mukaisesti. Moottorin pyörimisnopeuden laskiessa arvoon 200 rpm, tila muuttuu arvoon *Stopping* kuten taulukossa 1 on esitetty. Pyörimisnopeuden laskiessa takaisin arvoon 0 rpm, tila muuttuu arvoon *Ready* arvosta *Stopping*. Kun sytytysvirta kytketään pois päältä pyörimisnopeuden ollessa 0 rpm ja tilan ollessa arvossa *Ready*, tila muuttuu arvoon *Afterrun* ja pysyy siinä 30 s kunnes se muuttuu arvoon *Standby*. Verrattaessa tapahtumia kuvissa 14, 15 ja 16 sekä taulukoituja tapahtumia taulukossa 1 Agco Powerin määrittämään tilojen toimintalogiikkaan kuvissa 11, 12 ja 13 voidaan todeta luotu rajapinta Matlabin ja DLL:n välissä toimivaksi tämän työn vaatimusten osalta.

3.5 Simulaattorin integrointi osaksi moottorimallia

DLL:n sisältämän moottorinohjaimen SIL-simulaattorin integrointi osaksi Simulinkissä toteutettua dynaamista dieselmoottorimallia vaatii signaalien arvojen välittämistä moottorimallin ja ”Matlab funktio” -lohkon välillä. Lisäksi mallissa, johon SIL-simulaattori

integroidaan, on graafinen käyttöliittymä, jonka avulla ECU:lle annetaan haluttuja asetusarvoja esimerkiksi pyörimisnopeudesta tai kuormasta. ECU:lle syötetään myös erilaisia alkuarvoja tietyille signaaleille kuten ulkoinen ilmanpaine, polttoaineen ja öljyn lämpötila sekä akun varaus. Periaatekuva integroinnista on esitetty kuvassa 17.

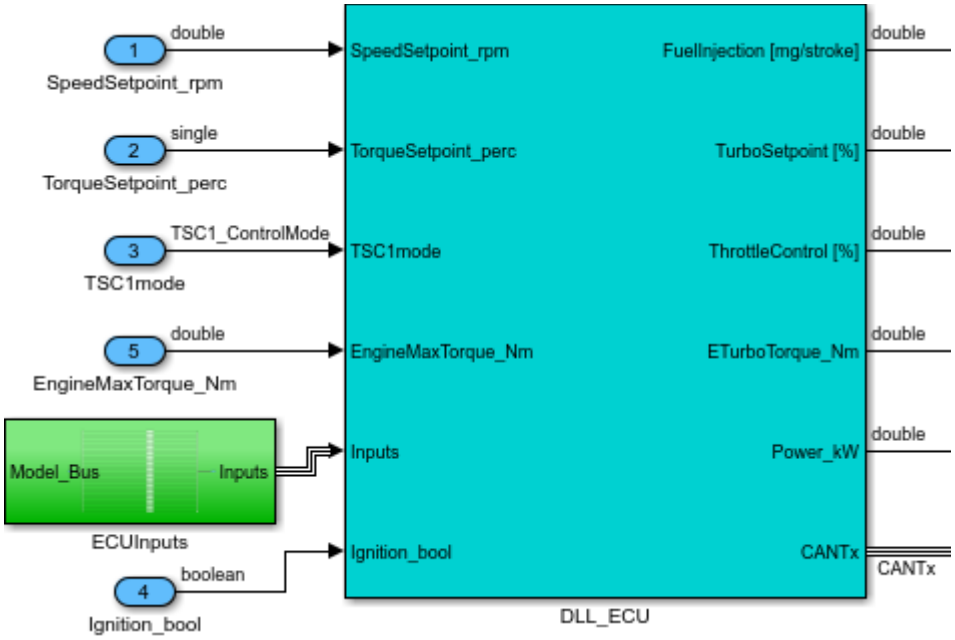


Kuva 17. *Periaatekuva toteutettavasta integraatiosta*

Moottorimallia ohjataan DLL:n SIL-simulaattorilta tulevilla signaalien arvoilla kuten kuvassa 17 on esitetty. Tärkeimpiä arvoja ovat ruiskutettava ainemäärä milligrammoina jokaista tahtia kohden, turboahtimen ohjauksen asetusarvo prosentteina sekä kaasuläpän ohjauksen ulostulo prosentteina. SIL-simulaattorille sisään tulevia signaaleita ovat moottorimallilta saatavat signaalien arvot, käyttäjän antamat asetusarvot graafisen käyttöliittymän kautta sekä alkuarvot.

SIL-simulaattorin integroinnissa käytetään hyväksi kappaleessa 3.3 esiteltyä ”Matlab funktio” -lohkoa. Lohko rakennetaan samaan Simulink-malliin moottorimallin kanssa, asetetaan herätesignaaleiksi kaikki kuvassa 17 esitetyt signaalityypit. Lohkon ulostuloiksi asetetaan kuvassa 17 esitetyt signaalit ”DLL ECU” -lohkolta moottorimallille. Sovelletaan luvussa 3.2 esitettyä “sil_set_value” -ja “sil_read_value” -funktiota signaalien arvojen asettamiseen ja lukemiseen.

Sisällytetään ”Matlab funktio” -lohko Simulinkin ”Subsystem”-lohkon sisään, jolloin mallista saadaan hierarkisempi ja yksinkertaisempi [12]. Kuvassa 18 on esitetty ”Subsystem” -lohko, jonka sisäänmenosignaalit ovat kuvassa 17 esitetyt lohkoon ”DLL ECU” sisään menevät signaalit. ”Subsystem” -lohkon ulostuloiksi määritetään SIL-simulaattorilta moottorimallille tulevat signaalit.



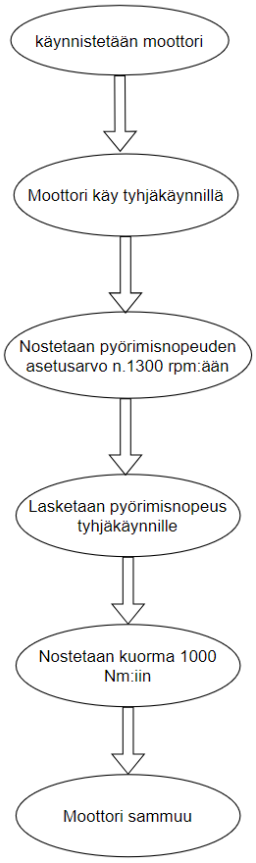
Kuva 18. *”DLL_ECU Subsystem” -lohko Simulinkissä*

Kuvassa 18 sisäänmenosignaalit 1–5 ovat graafisen käyttöliittymän kautta asetettuja asetusarvoja moottorimallille. Lohkoon sisään menevässä “Inputs”-signaalissa on koottu moottorimallilta tulevat signaalit Simulinkin ”Bus”-signaaliin. SIL-simulaattorilta tulevat ohjausarvot moottorimallille ovat lohkon vastesignaaleina. Ohjausarvoja ovat ruiskutettava ainemäärä (engl. *FuelInjection*), turboahtimen ohjauksen asetusarvo (engl. *TurboSetpoint*) sekä kaasuläpän ohjauksen ulostulo (engl. *ThrottleControl*). Lohkon ulostuloina ovat myös signaalit, jotka ilmaisevat sähköturbon pyydetyn väännön, moottorin hetkellisen kokonaistehon sekä Bus-signaalin mallinnetuista CAN-viesteistä, nämä signaalit eivät vaikuta tässä työssä vaadittavaan moottorimallin yksinkertaiseen toimintaan. Tässä työssä oletetaan, että moottorimalli on jo olemassa, joten sen tarkempi esittely on rajattu työn ulkopuolelle. Liitteen 1 (Moottorimalli Simulinkissä) kuvasta on rajattu pois kuvassa 18 esitetty ”DLL_ECU”-lohko, johon kuvassa oleva takaisinkytketty “Model_Bus”-signaali yhdistyy. Takaisinkytketyllä signaalilla välitetään arvoja moottorimallilta SIL-simulaattorille kuten kuvassa 17 on esitetty.

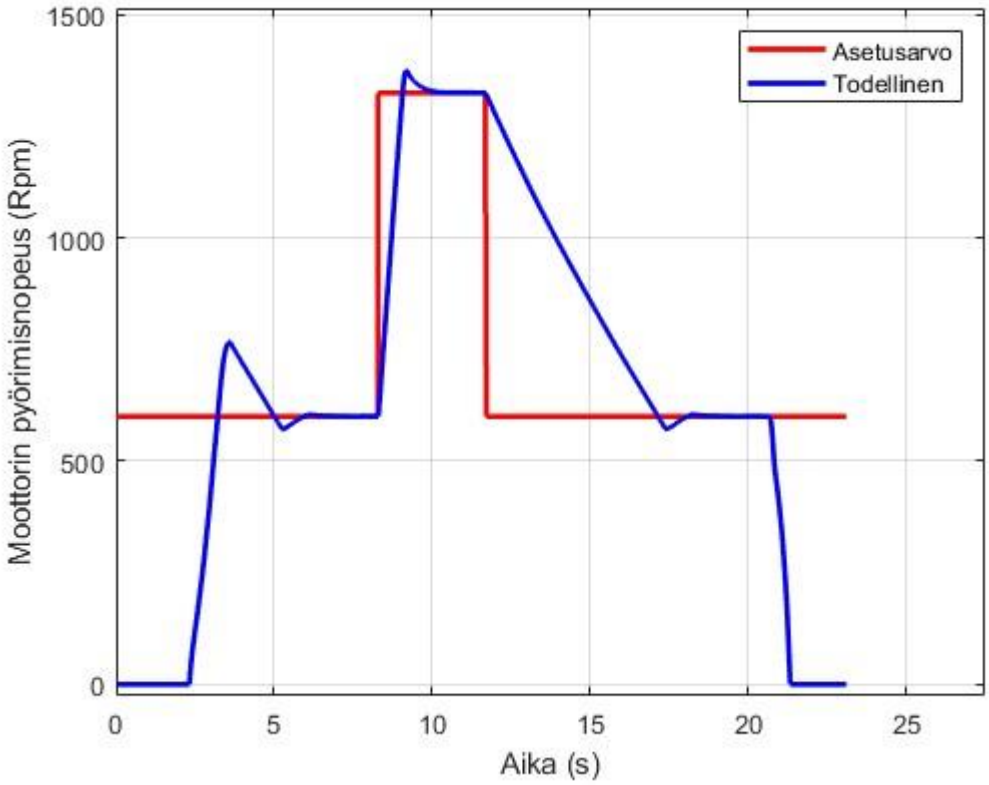
Kuvassa 18 esitetyn lohkon integrointi moottorimalliin toteutetaan yhdistämällä ”DLL_ECU”-lohkosta tulevat signaalit moottorimalliin siten, että signaali ruiskutettavasta ainemäärästä yhdistetään moottorimallin Engine-lohkon ”injQty”-signaaliin, turboahtimen asetusarvosignaali sekä kaasuläpän ohjauksen ulostulo signaali moottorimallin ”AirSystem”-lohkon ”TurboSetpoint” -ja ”ThrottleControl”-signaaleihin.

3.6 SIL-rajapinnan testaus moottorimallin kanssa

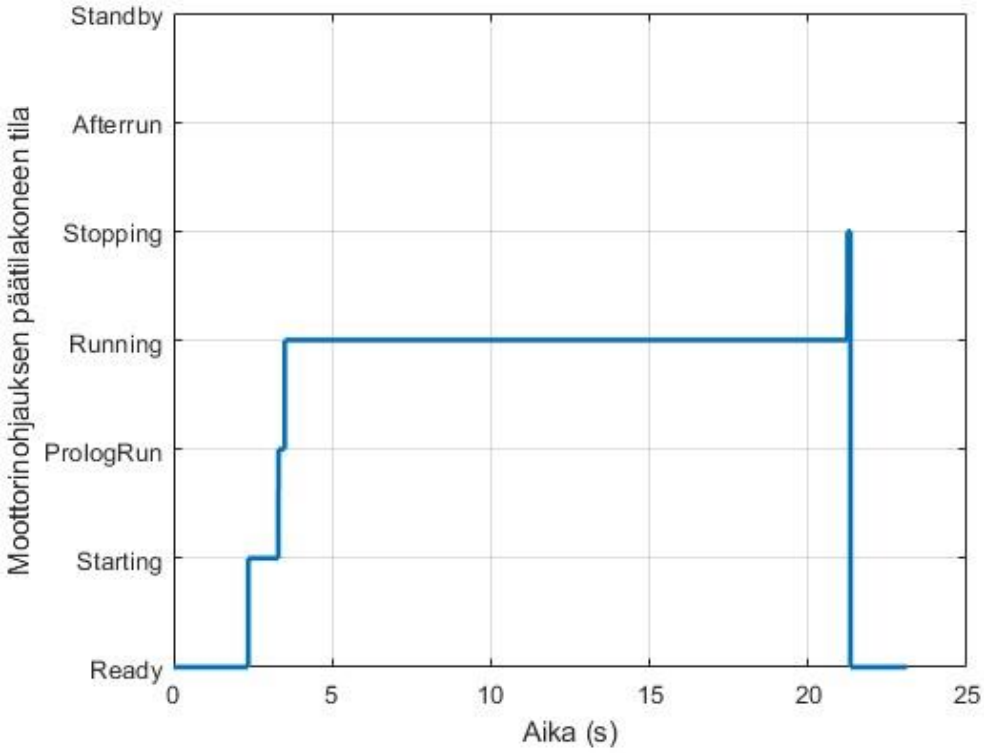
Suoritetaan rajapinnalle testi, jonka ensimmäisessä vaiheessa moottori käynnistetään. Toisessa vaiheessa asetetaan moottorinpyörimisnopeuden asetusarvo noin 1300 rpm:ään, verrattavissa kaasupolkimen painamiseen. Kolmannessa vaiheessa moottorin pyörimisnopeuden asetusarvo lasketaan 600 rpm:ään eli tyhjäkäynnille. Neljännessä vaiheessa moottorin kuorman asetusarvo asetetaan arvoon 1000 Nm, jolloin moottorin tulee sammua kuorman takia. Vuokaavio suoritettavasta testistä on esitetty kuvassa 19. Moottorin pyörimisnopeuden asetusarvo sekä todellinen arvo ajan funktiona kuvassa 20. Moottorinohjauksen päätilakoneen tila esitetään kuvassa 21 ajan funktiona sekä asetettu kuorma ajan funktiona kuvassa 22. Ajanhetket, tapahtumat sekä moottorinohjauksen päätilakoneen alku- ja lopputilat kyseisillä hetkillä ovat taulukoitu taulukossa 2.



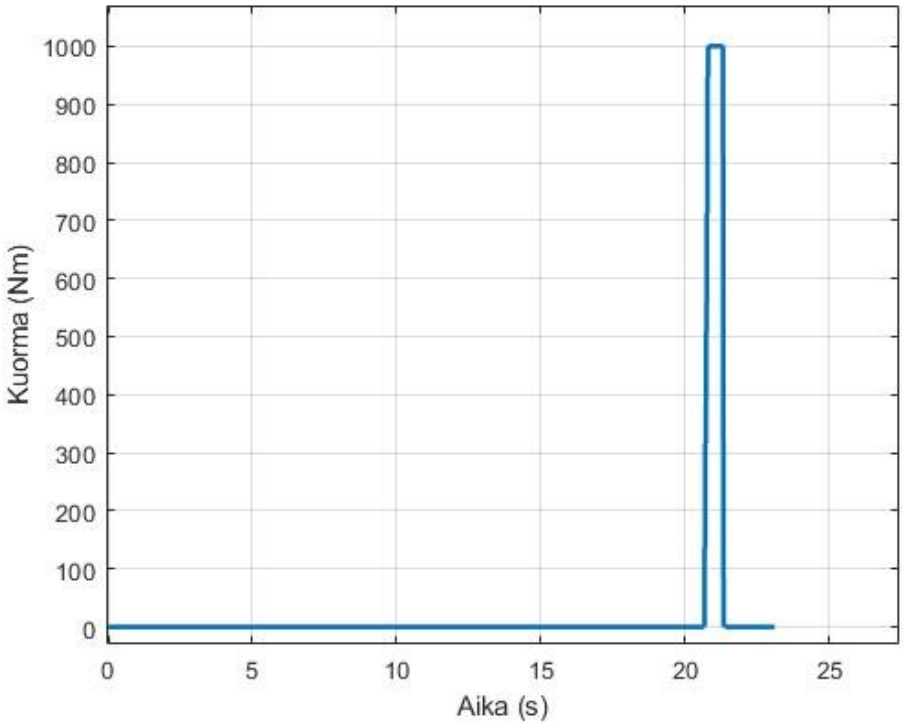
Kuva 19. *Testin vuokaavio*



Kuva 20. *Moottorin pyörimisnopeuden asetusarvo ja todellinen arvo ajan funktioina*



Kuva 21. *Moottorinohjauksen päätilakoneen tila ajan funktiona*



Kuva 22. *Asetettu kuorma ajan funktiona*

Taulukko 2: Testin vaiheet taulukoituna

Aika (s)	Alkutila	Lopputila	Tapahtuma
2,5	Ready	Running	Moottori käynnistetään
8	Running	Running	Pyörimisnopeuden asetusarvo nostetaan 1300 rpm:ään
9	Running	Running	Pyörimisnopeus saavuttaa asetusarvon 1300 rpm
12	Running	Running	Pyörimisnopeuden asetusarvo lasketaan 600 rpm:ään
17	Running	Running	Pyörimisnopeus saavuttaa asetusarvon 600 rpm
21	Running	Ready	Asetetaan kuorma 1000 Nm:iin, jolloin moottori sammuu

Testin ensimmäisessä vaiheessa moottori käynnistetään ja tila on arvossa *Ready*, jolloin kuvassa 20 moottorin pyörimisnopeus nousee ja asettuu ylityksen jälkeen asetusarvoonsa 600 rpm, samalla tila muuttuu tilojen *PrologRun* ja *Starting* kautta tilaan *Running,* mikä nähdään kuvassa 21. Toisessa vaiheessa pyörimisnopeuden asetusarvo nostetaan 1300 rpm:ään. Todellinen pyörimisnopeuden arvo nousee asetusarvoon ja asettuu siihen ylityksen jälkeen, mikä nähdään kuvassa 20. Vaiheessa kolme asetusarvo lasketaan 600 rpm:ään, jonka seurauksena kuvasta 21 nähdään, että todellinen

pyörimisnopeus laskee asetusarvoon 600 rpm. Testin neljännessä vaiheessa moottorin kuorma nostetaan 1000 Nm:iin kuten kuvassa 22 esitetään, tällöin moottorin pyörimisnopeus laskee nollaan rpm:ään eli moottori sammuu liian suuren kuorman takia, mikä nähdään kuvassa 21 tilan muutoksena tilasta *Running*, tilaan *Ready* tilan *Stopping* kautta.

Testin aikana huomataan kuinka moottorin pyörimisnopeuden asetusarvon asettamisen ja todellisen arvon muuttumisen välillä on moottorimallin dynamiikan aiheuttamaa hitautta kuvassa 20. Työn tavoitteena oli rakentaa toimiva rajapinta DLL:n sisältämän SIL-simulaattorin ja Matlabin välille niin, että SIL-simulaattori voidaan integroida ohjaamaan Simulinkissä olevaa dynaamista moottorimallia. Ajettaessa dynaamista moottorimallia SIL-simulaattorin ohjaamana kuvassa 19 esitetyn testin mukaan, voidaan kuvista 20, 21 ja 22 todeta DLL:n ja Matlabin välisen rajapinnan toimivan myös moottorimallin integroinnin jälkeen, koska moottorinohjauksen päätilakoneen tilat käyttäytyvät samalla tavalla kuin luvussa 3.4 esitetyssä HIL-nauhoituksessa.

4. TULOKSET

Tässä luvussa esitellään työssä toteutetun ratkaisun hyviä ja huonoja puolia verrattuna MIL-simulaatioon. Lisäksi tutkitaan ja esitellään ratkaisun hyötyjä osana tuotekehitystä.

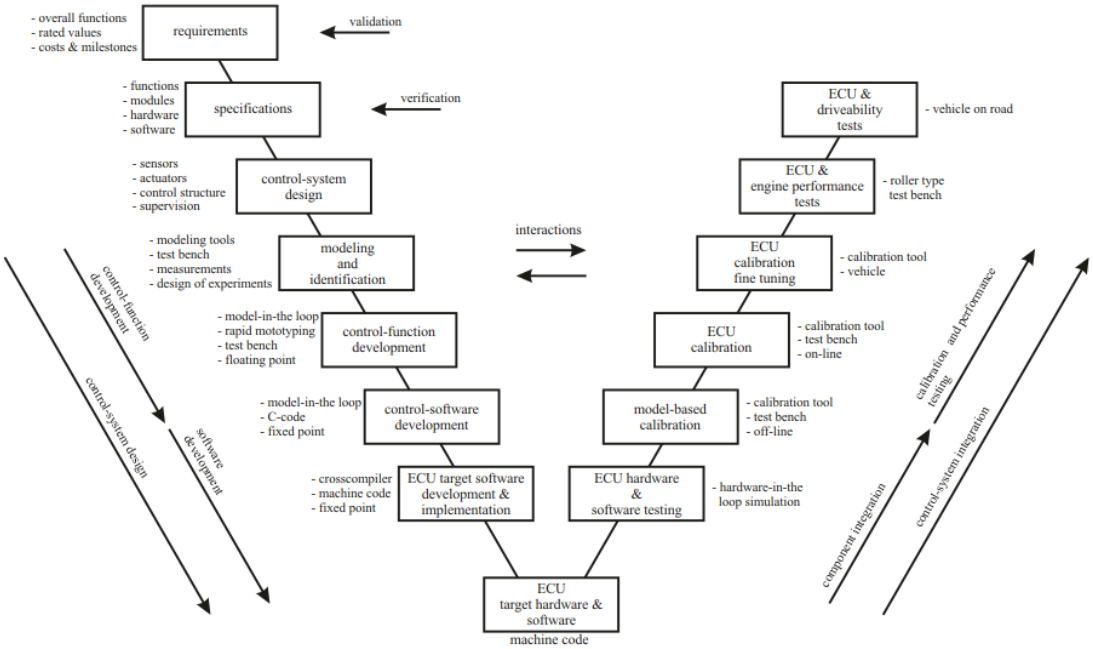
4.1 Vertailu

MIL-simulaatiossa dynaamista moottorimallia ohjaa Simulink-malleista koostuva moottorinohjaimen malli, jolloin kehittäjällä on testauksen ja kehityksen tukena vain Simulinkissä mallinnettu moottorinohjainohjelmisto. Tässä työssä rakennetun rajapinnan ja integraation avulla kehittäjä saa testauksen ja kehityksen tueksi myös suoraan C-kielellä toteutetut moottorinohjausohjelmiston ominaisuudet ja pystyy testaamaan sekä kehittämään niitä dynaamista moottorimallia hyödyntäen nopeammin omalla tietokoneellaan, HIL-simulaation sijaan. Toteutetun SIL-simulaattorin ja moottorimallin integraation hyötyjä tuotekehityksessä esitetään tarkemmin luvussa 4.2.

MIL-simulaatiossa pystytään näkemään konkreettisesti toteutetut logiikka- ja säätöratkaisut Simulink-mallien ansiosta, toisin kuin SIL-simulaattorissa, missä kehittäjä pystyy vain tarkastelemaan signaalien ja parametrien arvoja. MIL-simulaation avulla kehittäjä pystyy tekemään muutoksia Simulink-malleihin ja testaamaan niiden toimivuutta heti moottorimallin kanssa. Työssä toteutetun SIL-simulaattorin ja moottorimallin integraation tapauksessa kehittäjän on käännettävä Simulinkistä generoitu ohjelmakoodi DLL:ksi testatakseen toiminnallisuutta. MIL-simulaatiossa voidaan esittää suuri määrä signaalien ja parametrien arvoja simulaation aikana käyttäen hyväksi Simulinkin ”Port value label” –toiminnallisuutta [12]. SIL-simulaattorista signaalien ja parametrien arvoja voidaan lukea luvussa 3.2 esitetyllä ”sil_read_value” -funktiolla. Arvojen esittäminen Simulinkissä onnistuu luomalla ”Matlab funktio” -lohko halutuille signaaleille ja parametreille hyödyntäen ”sil_read_value” -funktiota.

4.2 Hyödyt tuotekehityksessä

Moottorinohjaimen tuotekehitysprosessia voidaan kuvata V-mallilla [1, s.323–328]. V-malli on esitetty kuvassa 23.



Kuva 23. *V-malli ECU:n kehityksessä [1, s.325]*

SIL-simulaatio testausta käytetään hyväksi, kun ohjelmisto on käännetty konekoodiksi ECU:a varten [1, s.323–328]. Tässä työssä toteutettua SIL-simulaattorin ja dynaamisen moottorimallin integraatiota voidaan käyttää hyväksi testaamaan käännettyä ohjelmistoa mallinnetun moottorin kanssa. Testaamista varten ohjausohjelmisto käännetään ECU:n ajoympäristön sijaan DLL:ksi yhdessä SIL-toiminnallisuuksien kanssa, jolloin DLL sisältää ECU:n ohjausohjelmiston ja sitä suorittavan SIL-ympäristön.

ECU:n ohjausohjelmisto sisältää sekä Simulinkillä mallinnettuja ja niistä generoituja ohjausfunktioita sekä suoraan C-kielellä toteutettuja ohjausfunktioita. Simulink-malleista generoitavia ohjausfunktioita voidaan testata MIL-simulaatiolla ennen generoimista C-kielelle ja edelleen kääntämistä konekielelle. C-kielellä toteutettuja ohjausfunktioita voidaan testata mallinnetulla moottorilla vasta HIL-simulaatiossa oikean ECU:n kanssa.

Työssä toteutetun järjestelmän avulla ohjausfunktioita, jotka toteutetaan C-kielellä, voidaan testata mallinnetulla moottorilla ilman oikeaa ECU:a ja HIL-järjestelmää. Myös Simulinkillä mallinnettuja funktioita, joiden toiminta riippuu C-kielellä toteutetuista funktioista, voidaan testata toteutetulla järjestelmällä. Toteutettu integraatio ei poista tarvetta HIL-simulaatiolle moottorinohjainohjelmiston tuotekehityksessä, mutta tietyissä tilanteissa voi vähentää sen käyttöä ja näin nopeuttaa ohjelmiston vaadittavaa testausta.

DLL:ksi käännetty ohjelmakoodi sisältää myös suuremman määrän parametreja ja signaaleja, kun Simulink-malleilla toimiva MIL-simulaatio. Tällöin voidaan kalibroida laajemmin haluttuja parametreja ja testata niiden toimintaa mallinnetulla moottorilla.

5. YHTEENVETO

Työssä toteutettiin rajapinta Matlabin ja DLL:n sisältämän moottorinohjaimen SIL-simulaattorin välille, jolloin SIL-simulaattori voitiin integroida ohjaamaan Simulinkissä valmiiksi mallinnettua dieselmoottorin dynamiikkamallia. Integroinnin jälkeen moottorimallia voidaan käyttää ECU:n SIL-simulaattorin vasteena jo käännetyn moottorinohjausohjelmiston testauksessa. Rajapinnan toteutuksessa käytettyjä luku –ja kirjoitusfunktioita voidaan käyttää myös simulaation aikana parametrien arvojen lukemiseen ja muokkaamiseen.

Työn teorialuvussa tutkittiin ja esiteltiin erilaisia mahdollisuuksia integroida C-kielellä toteutettuja ohjelmia Simulink-ympäristöön. Luvuissa 3.1.1 ja 3.1.2 esitellyt S-function builder ja C-caller soveltuvat pienempien ohjelmien tai yksittäisten funktioiden tuomiseen Simulink-ympäristöön.

Toteutettua rajapintaa testattiin sekä ilman moottorimallia, että moottorimallin kanssa. Molemmat testit antoivat tuloksen, jonka mukaan rajapinta toimii ilman dynamiikkaa tuottavaa mallia sekä mallin kanssa. Toteutettu integraatio ei poista tuotekehityksestä HIL-simulaation tarvetta, mutta tietyissä tilanteissa voi vähentää sen käyttöä ja nopeuttaa vaadittavaa testaamista.

Tässä työssä toteutettu rajapinta tukee DLL:n sisältämiä parametreja, jotka ovat skalaareita. Rajapintaa voidaan jatkokehittää lisäämällä tuki myös suurempia dimensioita omaavien parametrien lukuun ja muokkaamiseen. Kun tuki erityyppisille ja dimensioisille parametreille on tehty, voidaan erilaisten moottorityyppien ohjelmistojen kalibraatioita testata kehittäjän tietokoneella mallinnetun moottorin kanssa HIL-simulaation sijaan. Työtä voidaan jatkokehittää myös korvaamalla Simulinkin lohkojen välinen kommunikaatio Matlabin toteuttamilla virtuaalisilla CAN väylillä, joiden avulla kommunikaatiota ECU:n ja moottorimallin välillä voidaan kehittää vastaamaan enemmän reaalimaailmaa.

LÄHTEET

[1] R. Isermann, Engine Modeling and Control Modeling and Electronic Management of Internal Combustion Engines, TU Darmstad, 2014.

[2] K. Reif, Automotive Mechatronics, Automotive Networking, Driving Stability Systems, Electronics, Duale Hochschule Baden-Württemberg, 2015.

[3] S. Eshkabilov, Beginning MATLAB and Simulink, From Beginner to Pro [luku 1], second edition, APress Media, 2022.

[4] A. Mittal, R.N. Patel, Programming in MATLAB®, A Problem-Solving Approach, Pearson India, 2024.

[5] Microsoft, About Dynamic-Link Libraries. Saatavilla: https://learn.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries (23.7.2024)

[6] R. Allen, H. Klee, Simulation of Dynamic Systems with MATLAB and Simulink, 2nd Edition, CRC Press, 2016.

[7] Z. Dong, Z. Lihua, C. Yanbo, Research on Modeling of BLDCM Control System Based on S-function Builder [luku 3], IEEE, 2011.

[8] K. Reif, Diesel Engine Management, system and components, Duale Hochschule Baden-Württemberg, 2014.

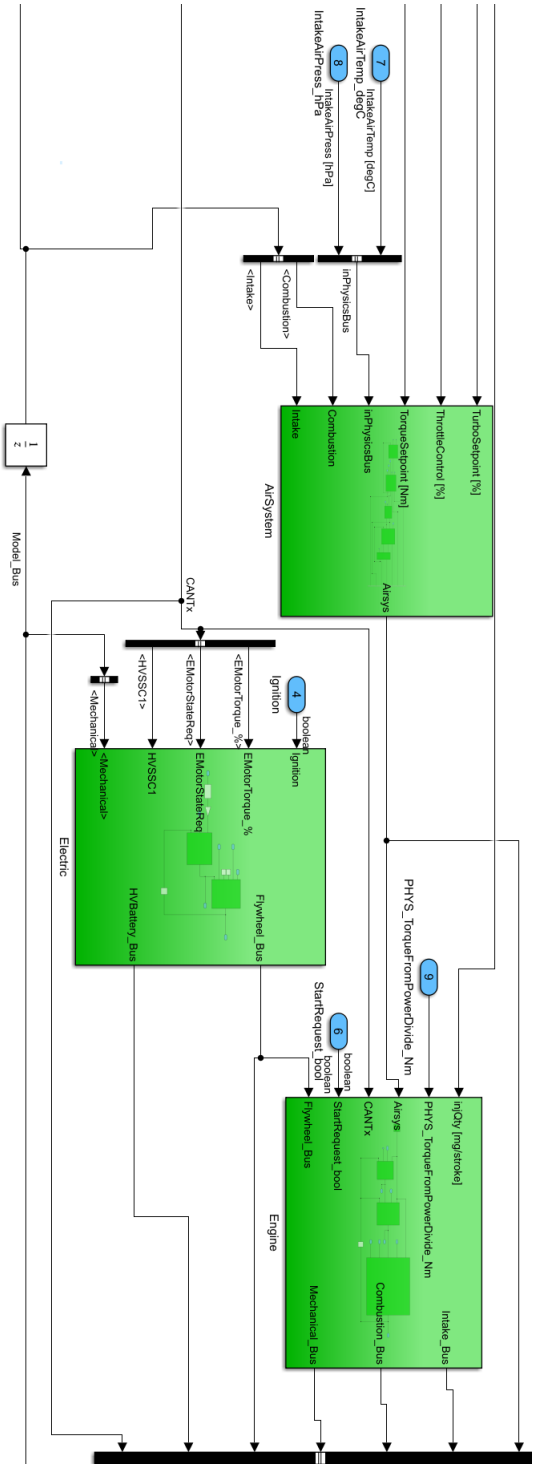
[9] A. Joshi, Automotive Applications of Hardware-in-the-Loop (HIL) Simulation, Ford Motor Company, SAE International, 2019.

[10] R.C. Seacord, Effective C, 10 program structure [luku 10], The IEEE Computer Society, 2020.

[11] M. Jormanainen, Sähköturbon ohjaus, Diplomityö, Tampereen yliopisto, 2019.

[12] Mathworks, Help Center. Saatavilla: https://se.mathworks.com/help (23.7.2024)

LIITE 1: SIMULINK MALLI



Kuva 1: *Moottorimalli Simulinkissä*. Airsystem-lohkossa on mallinnettu dieselmoottorin ilmajärjestelmä, Electric-lohkossa on mallinnettu hybridijärjestelmä ja Engine-lohkossa on mallinnettu dieselmoottori.

LIITE 2: MATLAB TOTEUTUS

```
function outputValue = sil_read_value(varname)

% Add the '.value' suffix to the variable name
varname = strcat(varname, '.value');

outputValue = double(evalin("base",varname));
end
```

***Kuva 1**. "sil_read_value" -funktion Matlab toteutus*

```
function Sil_set_value(varname, varvalue)
    % Set the value of the variable to the base workspace
    evalin("base", sprintf("%s.value = %d", varname, varvalue));
end
```

***Kuva 2**. "sil_set_value" -funktion Matlab toteutus*