Graz University of Technology

AVL List GmbH Graz

Kailin Tong

Investigations and testing of a hybrid control system using Model.CONNECT

Master Thesis

Graz University of Technology

Faculty of Mechanical Engineering and Economic Sciences

Institute of Automotive Engineering

Member of [FSI]

Supervisor:

Associate Prof. Dr. Mario Hirz, Institute of Automotive Engineering

Industry supervisor:

DI (FH) Simon Waltenberger, AVL List GmbH, Powertrain Controls

Graz, December 2017

Acknowledgement

Without all the help and support from my supervisors, colleagues and friends, the completion of my thesis work can’t be achieved. Here in this acknowledgement, it is of great significance to express my gratitude to them.

I sincerely appreciate Prof. Mario Hirz, for being my supervisor at Graz University of Technology. When I was faced with challenges, he gave me helpful support and inspiring ideas. He also helped me to organize the workflow of my final exam and refined my thesis with patience and excellent scientific literacy.

I owe a deep sense of gratitude to my supervisor at AVL, Simon Waltenberger, who continuously supported my research topic with countless efforts and valuable advice, no matter it is basic knowledge of the project, practical problems that I met, or literature writing technique, etc. And he gave me much motivation and encouragement throughout my thesis work. His optimistic attitude and extremely patient dedication always inspires me to accomplish my task with confidence.

Many thanks to Markus Kogler at AVL, who supported me with the project at the beginning, helped me get to know different software and hardware. Thanks to Kun Wang at AVL, who helped me solve a number of practical problems. Thanks to Norbert Thek at Virtual Vehicle and Günter Lang at AVL for supporting me with Model.CONNECT. Thanks to Nivedhitha Murthy, Bhargav Adabala, Harald Brandl and Benny Locker at AVL for answering my questions during the thesis work.

Thanks to my friends, Liang Shao, Yuanfeng Wang and Zhenyue Hu, who gave me suggestions about my thesis. Thanks to Min Liu and Zecheng Zhang, a kind and nice family, for always taking care of my life. A particular thank goes to my family and my girlfriend, Hang Liu, who encouraged and inspired me to find a solution to tough problems.

There are also some friends and colleagues whose names are not showed here, but they are important persons to me, I will keep their kindness in mind.

Kailin Tong

December 2017, Graz

Statutory Declaration

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ……………………………………………………………………………….

(Unterschrift)

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and I have explicitly marked all material, which has been quoted either literally or by content from used sources.

………………………….. ……………………………………………………………

(Date) (Signature)

Abstract

Due to the growing complexity of vehicles, especially hybrid vehicles, comprehensive ECU tests at the early stage of development will be increasingly necessary. One solution is the so-called co-simulation, which not only enables co-simulation between various tools from different domains, but also makes the connection between virtual models and real hardware possible. Thus, there arise numerous benefits, e.g. front loading, saving of costs and time, distributed calculation, flexibility and scalability. In this thesis, three kinds of tests using the co-simulation platform Model.CONNECT are investigated. These tests include TCU (Transmission Control Unit) virtual test, multi-ECU (Electric Control Unit) virtual test and HVCU (Hybrid Vehicle Control Unit) real test.

In the TCU virtual test, the calculation of TCU model and plant model is distributed into two MATALAB workspaces by applying co-simulation. The effect of co-simulation on the test results is investigated. Co-simulation always brings extrapolated errors. Additionally, sequential co-simulation with similar sequence as the original simulation has the least overall errors. Contrary to expectation, in this work, the sequential or parallel co-simulation has little influence on the overall simulation time and the overall computation speed is not enhanced by co-simulation.

In the multi-ECU virtual test, a co-simulation platform comprising HCU, TCU and newly merged plant model is created. By the means of co-simulation, the proposed test platform is used to analyze the operation of the hybrid control system. Also, the proposed platform can be integrated into the existing tool chains. The setup is finished but the simulated vehicle in the plant model cannot really move. The most possible reason is wrong calibration of control units； however, this has not been further investigated in this work

In the HVCU real test, the basic test platform is developed, optimized and extended. The proposed test system empowers the in-office HiL testing with low cost configurations. Seven criteria are proposed to evaluate it. The proposed test system is applicable for different test cases and can partly reflect the behavior of a real car. However, it is a soft-real time system and cannot guarantee the repeatability of test results. Moreover, it has other merits including ease of testing, possibilities for extension and potential of ease of setup. The solutions to the challenges are also summarized.

Key words: co-simulation, real-time, HiL, hybrid powertrain system, Model.CONNECT

Kurzfassung

Aufgrund der zunehmenden Komplexität von Fahrzeugen, insbesondere Hybridfahrzeugen, sind umfassende Steuergeräte-Tests in einem frühen Stadium der Entwicklung erforderlich. Eine Lösung ist die sogenannte Co-Simulation, die nicht nur eine Co-Simulation zwischen verschiedenen Werkzeugen aus verschiedenen Domänen, sondern auch die Verbindung zwischen virtuellen Modellen und realer Hardware ermöglicht. Daraus ergeben sich zahlreiche Vorteile wie Frontloading, Kosten- und Zeiteinsparung, verteilte Berechnung, Flexibilität und Skalierbarkeit. In dieser Arbeit werden drei Arten von Tests unter Verwendung der Co-Simulationsplattform Model.CONNECT untersucht. Diese Tests umfassen einen virtuellen TCU-Test (Transmission Control Unit), einen virtuellen Test mit mehreren ECUs (Electric Control Unit) und einen echten HVCU-Test (Hybrid Vehicle Control Unit).

Im virtuellen TCU-Test wird die Berechnung von TCU-Modell und Anlagenmodell in zwei MATLAB-Workspaces durch Co-Simulation verteilt. Mittels Co-Simulation wird die vorgeschlagene Testplattform zur Analyse des Betriebs des Hybridsteuerungssystems verwendet. Die Auswirkung der Co-Simulation auf die Testergebnisse wird untersucht. Die Co-Simulation bringt immer extrapolierte Fehler. Zusätzlich weist die sequentielle Co-Simulation mit ähnlicher Sequenz wie die ursprüngliche Simulation die wenigsten Gesamtfehler auf. Entgegen der Erwartung hat die sequentielle oder parallele Co-Simulation in dieser Arbeit wenig Einfluss auf die gesamte Simulationszeit und die Gesamtberechnungsgeschwindigkeit wird durch die Co-Simulation nicht verbessert.

Im virtuellen Multi-ECU-Test wird eine Co-Simulationsplattform mit HCU, TCU und neu zusammengeführtem Anlagenmodell erstellt. Mittels Co-Simulation wird die vorgeschlagene Testplattform verwendet, um den Betrieb des Hybridsteuerungssystems zu analysieren. Auch die vorgeschlagene Plattform kann in die bestehenden Werkzeugketten integriert werden. Das Setup ist beendet, aber das simulierte Fahrzeug im Anlagenmodell kann sich nicht wirklich bewegen. Der wahrscheinlichste Grund ist eine falsche Kalibrierung der Steuergeräte; dies wurde jedoch in der vorliegenden Arbeit nicht weiter untersucht.

Im HVCU-Real-Test wird die grundlegende Testplattform entwickelt, optimiert und erweitert. Das vorgeschlagene Testsystem ermöglicht das HiL-Testen im Büro mit kostengünstigen Konfigurationen. Zur Bewertung werden sieben Kriterien vorgeschlagen. Das vorgeschlagene Testsystem ist für verschiedene Testfälle anwendbar und kann teilweise das Verhalten eines realen Autos widerspiegeln. Es ist jedoch ein Soft-Real-Time-System und kann die Wiederholbarkeit der Testergebnisse nicht garantieren. Es hat jedoch andere Vorzüge, einschließlich der Einfachheit des Testens, der Möglichkeiten für die Erweiterung und des Potentials für die Leichtigkeit des Aufbaus. Die erarbeiteten Lösungsansätze für die Herausforderungen sind ebenfalls zusammengefasst.

Schlüsselwörter: Co-Simulation, Echtzeit, HiL, Hybrid-Antriebssystem, Model.CONNECT

Contents

Abbreviations

Abbreviation	Meaning
ABS	Anti-lock Braking System
AC	Alternative Current
ADAS	Advanced Driver Assistance Systems
ADD	Automotive Data Dictionary
BMS	Battery Management System
BN	Boardnet (12V)
BSW	Basic Software
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CIL	Custom Interface Layer
CSV	Comma-Separated Values
CMD	Command
CPU	Central Processing Unit
CV	Conventional Vehicle
DBC	Data Base CAN
DC	Direct Current
DLL	Dynamic Link Library
ECU	Electronic Control Unit
EMS	Engine Management System
EPB	Electric Parking Brake
ESP	Electronic Stability Program
EV	Electric Vehicle
GUI	Graphical User Interface
HCU	Hybrid Control Unit

Abbreviation	**Meaning**
HEV	Hybrid Electric Vehicle
HiL	Hardware in the Loop
HV	High voltage
HVAC	Heating, Ventilation and Air Conditioning
HVCU	Hybrid Vehicle Control Unit
HVIL	High Voltage Interlock
HW	Hardware
HYB	Hybrid
ICEV	Internal Combustion Engine Vehicle
ICOS	Independent Co-Simulation
ID	Identification
JMS	Job Management System
LIN	Local Interconnect Network
MC	Model.CONNECT
MCU	Motor Control Unit
MiL	Model-in-the-Loop
MOST	Media Oriented Systems Transport
MSE	Mean Square Error
NEPCE	Nearly Energy Preserving Coupling Element
NTC	Negative Temperature Coefficient
OS	Operating System
OBD	On Board Diagnosis
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCU	E-Motor Control Unit (Power Control Unit)
PT	Powertrain
PVT	Private

Abbreviation	Meaning
PWM	Pulse Width Modulation
RAM	Random-Access Memory
RT	Real-Time
RTOS	Real-Time Operating System
SiL	Software in the Loop
SLC	Shift Lever Controller (Gear Lever)
SOC	State of Charge
SRS	Airbag Module
SW	Software
TCU	Transmission Control Unit
USB	Universal Serial Bus
XCP	Universal Measurement and Calibration Protocol

Symbol directory

Symbol	Meaning	Unit
U	Voltage	V
I	Current	A
R	Resistance	Ω
C	Capacity	C
p	Hydraulic Pressure	bar
f_c	Cut-off Frequency	Hz
δt	Micro Time Step	s
ΔT	Macro Time Step	s

1 Introduction

Today, the problem of vehicle emissions associated with depletion of petroleum resources has become a main concern of automotive industry, and it is commonsense widely that technologies applied in contemporary automobiles should be adapted or replaced for the future. In order to obtain significant cut in greenhouse gases (GHG) and oil utilization, there is a variety of different technologies [1]-[2]. So far, one of the most hopeful technologies is the hybrid electric vehicle (HEV) as it improves fuel economy and meets the demands of customers as well, especially those who desire to drive in long ranges [3]. Thus, it presents a practical alternative to the conventional vehicle (CV) in the near future. A typical HEV integrates the conservative internal combustion engine (ICE) as its primary power with batteries/electric motors as energy buffer [4].

Meanwhile, the quantity of control units in the vehicle has climbed considerably since electronics was introduced in vehicles at the beginning of the 1980s [5]. Growing number of functions have been provided for the safety and comfort of the driver, from the control and monitoring of the powertrain, safety-relevant tasks such as ABS or ESP, to navigation systems and networking of mobile phones. Particularly, HEVs are more complex than conventional vehicles, which comprise complicated interactions between different powertrain devices and incorporate complicated electronic control unit (ECU) network [6]. Furthermore, for the vehicles in future, comprehensive ECU tests will be more necessary than ever before, with rising complexity and extent of software at a breathtaking pace [7]. The wide recognition of V-cycle model in automotive industry also brings more efforts of testing and integrates offline controller development with hardware-in-the-loop (HiL) testing at the final stage of the development [4]. Thus, the development of comprehensive ECU testing is extremely needful and presents a tremendous challenge for HEV manufacturers. One approach to improve development processes is using the so-called co-simulation. It not only supports co-simulation between various offline control unit models, plant models and supplementary models, but also provides an environment where offline models and real-time devices can be integrated.

1.1 Motivation

The individual ECU software and hardware prototype is normally delivered by various teams. To reduce errors caused by distributed development, one solution is the implementation of integrated functional tests of multiple ECU models at the early phase of development. With the assistance of Model.CONNECT, multiple ECUs can be integrated and tested in a co-simulation environment during the model-in-the-loop (MiL) phase. Therefore, problems resulting from the communication between ECUs can be resolved with comparatively minimal efforts. Another benefit arising from the co-simulation is that its simulation speed can be accelerated due to the distributed calculation of models.

Moreover, by the application of co-simulation between models and real-time devices, some parts of HiL testing jobs can be accomplished in office with low-cost configurations rather than in a test bed or a real vehicle. This can decrease the development time as well as the expense of testing and diminish the cost of late renovation of the design. Also, the flexibility and scalability of the co-simulation can cater to various test demands.

Finally, not only in the field of automotive industries - the huge demand for low-cost and flexible HiL solutions also exists in the field of aerospace, robotics, power systems, national defense, etc.

1.2 Tasks

There are three main tasks in this thesis.

1. Setup and investigations of the Transmission Control Unit (TCU) virtual test environment with Model.CONNECT

The simulation of TCU model is extended by the co-simulation program Model.CONNECT. The data exchange between the TCU and the plant model no longer takes place in one Simulink of one MATLAB workspace, but in two MATLAB workspaces with data transmitted by MC. This test arrangement is used to validate the test arrangement as well as workflow, and compare the running time with original tests. Moreover, the influence of Model.CONNECT on the simulation results and the choice of co-simulation sequence shall be investigated.

2. Setup and investigations of Multi-ECU (Hybrid Control Unit + Transmission Control Unit) virtual test environment with Model.CONNECT

Similar to the first task, the simulation of the single TCU model as well as the single HCU model are extended by Model.CONNECT. With the help of Model.CONNECT, the data exchange between the TCU model, the HCU model and the plant model in different MATLAB workspaces can be achieved and the final test result of the co-simulation shall be investigated. Another target is the attempt of integrating Model.CONNECT into the existing control software development procedure and tool chains. Therefore, it could be applied in future practical projects.

3. Improvement and extension to the basic HVCU real test platform and exploration of an efficient workflow for further projects

A basic platform of the HVCU real test was established, however, until the work started, the original platform could not produce any meaningful result. Thereby the first step is to make the test platform really run. Furthermore, many aspects concerning the developed test system shall be assessed according to different criteria. As this work is an initiative application of Model.CONNECT in the field of hybrid control system HiL testing, the optimal workflow, configurations and the methods of evaluation are going to be investigated.

1.3 Main content of the thesis

The main contents of this thesis is divided into the following parts:

In chapter 1, the motivation and tasks of this thesis are briefly described.

In chapter 2, the background knowledge about hybrid vehicles, bus systems, co-simulation, real-time co-simulation, operating systems and jitter is introduced.

In chapter 3, the used programs in this work are presented and a description of the functions used in this thesis is given.

In chapter 4, the test arrangement as well as setup of the TCU virtual test is described. The test results and computation time are analyzed and compared with the original test.

In chapter 5, the test arrangement as well as setup of the Multi-ECU virtual test is described. And the workflow for it and how to integrate it into the existing tool chains is elaborated. Finally, the test results are analyzed.

In chapter 6, firstly an introduction to the basic HVCU real test platform is given. The improvements to the basic platform in light of I/O simulation, real-time performance and other changes in the model are elaborated. An optimized workflow, test specification and test sequence is also shown in this chapter. Many test cases are implemented and seven criteria are proposed to evaluate the developed test system, which clearly shows its advantages, potentials and problems.

In chapter 7, the work of this thesis is summarized and the future work is exhibited.

2 Literature review

2.1 Overview of hybrid vehicles

Hybrid electrical vehicles (HEVs) are the focus of many research interests because of their improved performance and longer operating time. An HEV is defined as a vehicle with at least two different propulsion energy sources and at least one of them is able to deliver electrical energy. In addition, an HEV electric powertrain takes advantage of bidirectional power flow in order to recapture the heat losses in braking [2].

This technology enables the vehicle to acquire the benefits from different energy sources. The three main potential edges of a HEV are summarized as follows [8]:

1. It is possible to recovery and store the energy normally lost during braking events for a later use.

2. The downsizing of the primary engine can be achieved.

3. Optimization process and higher efficiency is more available due to the fact that primary power source operates under a more constant load.

Depending on the configuration of the drive train, hybrid vehicles are divided into three basic types and one additional type, which represents some newly introduced HEVs: series hybrid, parallel hybrid, series-parallel hybrid and complex hybrid [2]-[3],[8]-[9] (Figure 1).

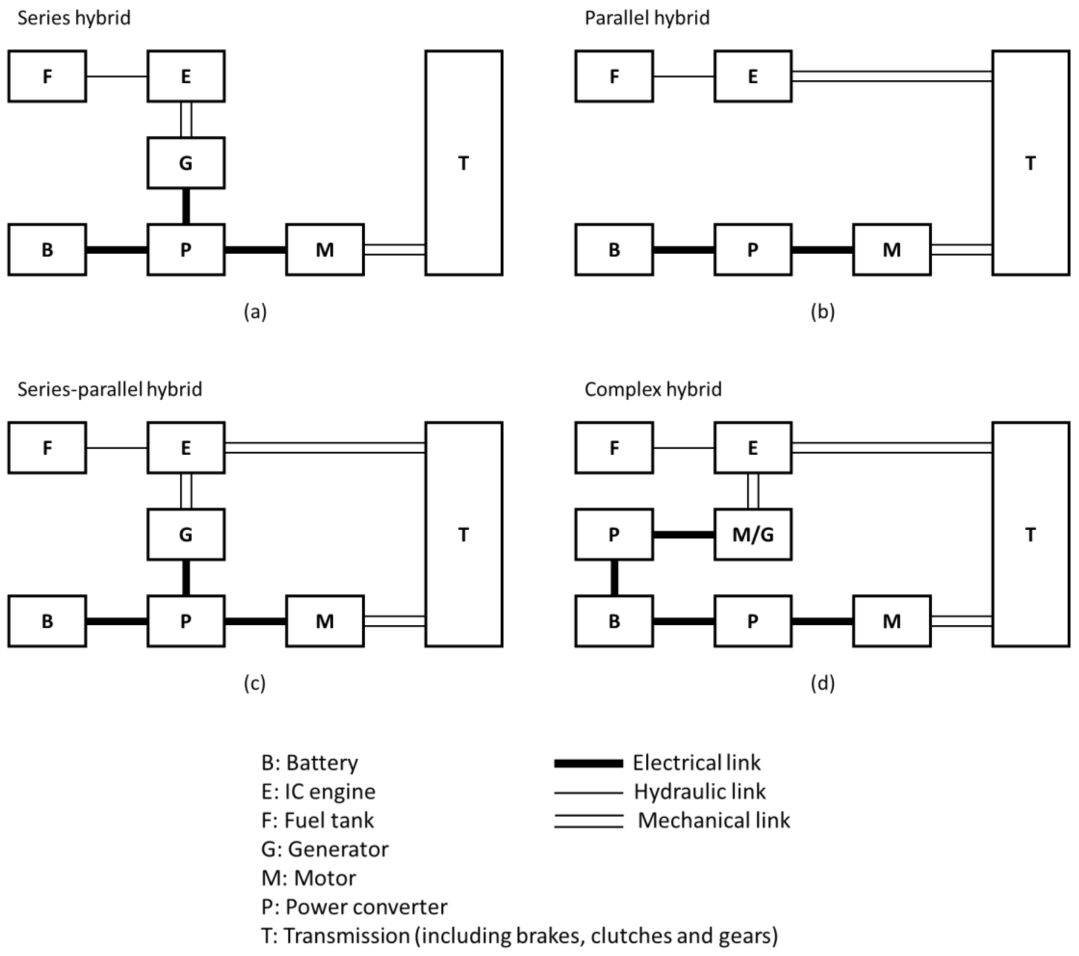


Figure 1: Schematics of different types of HEV drive train configurations [3]

(a) Series hybrid. (b) Parallel hybrid. (c) Series–parallel hybrid. (d) Complex hybrid.

2.1.1 Series hybrid system

In a series hybrid vehicle, as shown in Figure 1(a) the internal combustion engine (ICE) has no mechanical connection to driving wheels. The electricity converted by the generator from the ICE's mechanical output either charges the battery, or bypasses the battery and supplies the propulsion of wheels. Theoretically, it can be viewed as an ICE-assisted EV that is targeted at extension of the driving range in comparison with the electric vehicle (EV). The ICE can work at its optimal operation point, which leads to a fuel saving and maintains the battery charge regardless of the driving condition. The main disadvantage lies in the comparatively low efficiency due to the complex energy flow. Another problem of a series hybrid system is the size of all propulsion devices because of their dimensioning for the case of high power demand.

2.1.2 Parallel hybrid system

In contrast to the series hybrid, both the ICE and electric motor in a parallel hybrid system (Figure 1(b)) are allowed to deliver the power in parallel to propel wheels. Conceptually, Parallel hybrid system is an electric-assisted ICEV (Internal Combustion Engine Vehicle) for the purpose of reducing emissions and improving fuel economy. This kind of configuration provides freedom to select a combination of traction sources. In the conventional configuration, the propulsion might be supplied by a single ICE, a single electric motor or by both. Therefore, a smaller engine can be used. Furthermore, a parallel HEV configuration might require a relatively smaller battery capacity compared to a series HEV, which helps to lower the mass of drive train. Another advantage is that there are less energy conversion stages in a parallel hybrid powertrain compared to a series hybrid powertrain, which brings higher overall efficiency. However, it might show disadvantages in urban driving in case that the ICE operates in a low-efficiency and high-emissions range.

2.1.3 Series-parallel system

This configuration incorporates the features of a series HEV and a parallel HEV (Figure 1(c)), by adding a mechanical link in contrast to the series HEV and adding a generator in comparison to the parallel HEV. In spite of possessing merits of both series and parallel HEVs, the main problem of this system is its higher complexity and costs.

2.1.4 Complex hybrid system

This system (Figure 1(d)) comprises a more complicated configuration that is different from the previous three kinds. It seems to be similar to the series-parallel system. However, the significant difference results from the unidirectional energy flow of the generator in the series-parallel hybrid system and the bidirectional energy flow of the electric motor in the complex hybrid system. The bidirectional energy flow makes versatile operating methods possible, especially the three-propulsion-power (two electric motors and one ICE) operating method. By contrast, this kind of operating method cannot be provided by the series-parallel hybrid system. However, like the series-parallel hybrid system, this system is relatively complicated and costly. Nevertheless, this system is adopted in some new models of HEV for dual-axle propulsion.

2.2 Bus systems in vehicles

Electrical and electronic components in modern vehicles normally influence and complement each other. With the increasing demands for the volume of data that is being exchanged, one solution is the development of a serial bus system. A bus system is a communication network in which data from several sources can be transferred [10]. The most commonly used bus systems in the vehicle are summarized as follows.

2.2.1 CAN bus

The CAN bus (Controller Area Network) was initially in 1991 applied in a motor vehicle in mass production and since then it has established itself as a standard bus system in passenger and commercial vehicles [10]. It defines a standard and reliable communication between components in a vehicle including controllers, actuators, sensors and other nodes in real-time applications. Also, its application field is being extended to many other fields like aerospace, robots and electricity industry [11]. In light of the controllers in the vehicle, CAN bus communication is extremely important since usually several CAN buses are used for one controller. For example, a hybrid control device can communicate with the transmission control device via one CAN bus, with the battery control device via a second CAN bus and with the power electronics of the DC/DC converter via a third one.

The CAN bus is utilized in different domains in a vehicle where the demand for the network is diversified. To achieve an optimal benefit over cost, the CAN bus is split up into high-speed and low-speed CAN bus. The high-speed CAN (CAN-C) performs at bit rates from 125 kBit/s to 1 MBit/s. As a result, the data transmission can be capable of fulfilling the real-time requirements of the powertrain. By contrast, low-speed CAN operates at a bit rate from 5 to 125 kBit/s. It is mainly used in the field of comfort/convenience where the requirement of real-time data transfer is relatively lower [10].

A network node (Figure 2) is responsible for the data transmission. It contains a microcontroller for the application software, a CAN controller as well as the CAN transceiver (Bus driver). The CAN controller is designed for transferring and receiving data. The bit stream is firstly generated by CAN controller according to the binary data to be transferred, after which it is forwarded to the transceiver on the TxD line. This processes signals and the desired voltage used for differential data transfer is thereby generated. Finally, the CAN transceiver sends the processed bit stream on the CAN bus line with CAN_H and CAN_L [10].

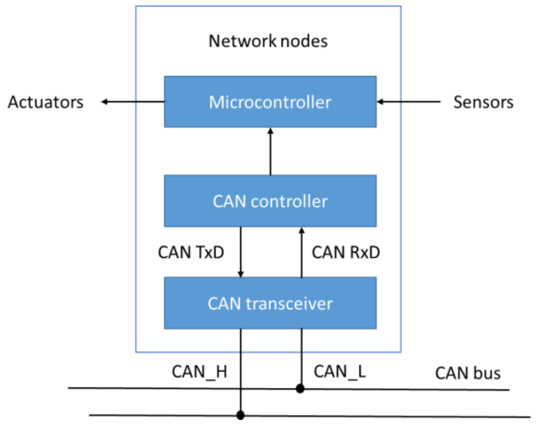
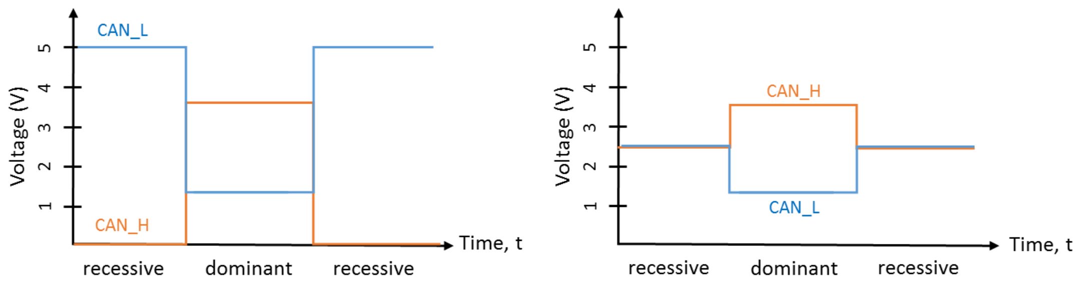


Figure 2: Network nodes in the CAN [10]

The CAN bus makes use of two states for communication: dominant states (binary “0”) and recessive states (binary “1”). When CAN transceiver receives messages, the signal level is converted to logical states. More specifically, the CAN_L is subtracted from the CAN_H by a differential amplifier. This differential data transfer makes it possible to filter out disturbance pulses on the bus line if lines are twisted. The voltage level and the corresponding states for high-speed and low-speed CANs are illustrated in Figure 3 [10].



(a) Voltage level of the low-speed CAN (b) Voltage level of the high-speed CAN

Figure 3: CAN bus signal levels [10]

The design of CAN bus brings numerous benefits. The CAN bus arrangement significantly lessens the required quantity of connections between nodes compared with point-to-point structure. Meanwhile, the cost of the bus system as well as propagation delays can be reduced [12]. Furthermore, many features of CAN bus including high reliability of data transfer, flexibility of configuration, non-destructive bus-accesses method and so on, make it a popular application in automotive industry. However, one limitation of CAN is that it lacks deterministic scheduling for real-time events since message routing theoretically can be delayed by message arbitration [10],[13].

2.2.2 LIN bus

The LIN (Local Interconnect Network) bus was developed by several manufacturers in 1998 to provide a simpler, more cost-effective alternative fieldbus technology for low bit rates (up to 20 kBit/s) [14]. A master-slave method is applied. The master, which is normally an electronic control unit (ECU) connected with a superordinate bus system, gives task to slaves (Figure 4). Communication on the LIN bus is implemented time-synchronously and the time step is defined by the master. As a result, a strictly deterministic LIN bus response is obtained.

One example of LIN networks as a subordinate bus in the domain of roof/wiper is given in Figure 4. In the example, a central ECU works as the master and there are four slaves: wiper actuator, rain/light sensor, garage-door opener and mirror. The master is also a gateway to the Diagnosis CAN bus, Body CAN bus as well as Chassis CAN bus [10].

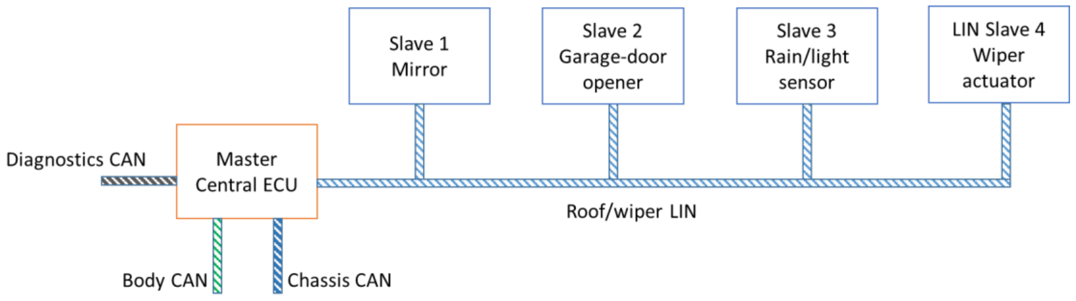


Figure 4: LIN bus with master and slave nodes [10]

LIN bus is usually applicable to low-performance systems. One obvious advantage of LIN bus lies in the cost-effectiveness of sufficient functions in the application area.

2.2.3 MOST bus

The MOST bus (Media Oriented Systems Transport) is a high-speed multimedia network technology developed for networking of infotainment components in a vehicle. Besides traditional entertainment components (like CD players), infotainment components also consist of video functions, access to mobile communications as well as a route guidance system. These functions set a higher demand for the bus system since they require a high bit rate and a synchronization of the data transfer.

In light of the transmission of data, the MOST bus sustains following channels [10]:

- The control channel, which is utilized for the simple transmission of control demands, for the statues of signaling device and for the system-management functions
- Flexible number of synchronous channels that are capable of carrying audio as well as video data
- Asynchronous channel, on which data is sent in packets without a fixed data rate

2.2.4 FlexRay

The FlexRay consortium was founded in 1999 to meet the increasing demands on bandwidth as well as transmission security and began to compile requirements for the new FlexRay system. It is a deterministic and fault-tolerant bus, which still offers possibilities for a future expansion. The main application fields of the FlexRay are drive train systems and active safety systems with x-by-wire. Nevertheless, it could also be utilized in the areas of passive safety, comfort/convenience and body electronics [10].

FlexRay takes advantage of two different types of bus access. Considering deterministic transfer properties, it provides time-controlled bus access. For those applications, which have less demands on the deterministic transfer properties, FlexRay attempts to make an effective use of the available capacity of transfer. The cyclic communication helps to combine these two approaches. The communication cycle contains a dynamic and static segment. With the former, a priority-controlled data transmission is possible. With the static segment, the transfer property is able to be deterministic. Therefore, both synchronous (real-time) and asynchronous data transfer is available in case of different demands from various systems in the vehicle [10],[15]-[16].

The redundant transmission of messages is possible since the FlexRay is executed with two transmission channels. The high speed of data transmission for FlexRay is an apparent merit. The FlexRay bus specifies three standard bit rates: 10 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s. Without redundant transmission, it can be as high as 20 Mbit/s. Compared with CAN, FlexRay also offers additional reliability features. Specifically, the redundant communication capability allows for duplication of network configurations as well as schedule monitoring by hardware [15]-[16].

2.3 Co-simulation

In automotive industry, greater importance has been attached to the simulation of entire systems or subsystems of a vehicle. Due to the increasing complexity of the vehicle, various technical disciplines as well as manifold modeling environments have been developed for specific problems and different domains. The task of a co-simulation environment is to generate a heterogeneous simulation environment, in which the data exchange between different simulation tools can be implemented and the influence of the integrated system can be assessed [17]. Moreover, since a classical co-simulation is no longer able to solve all occurring problems in the vehicle development procedure, one extension is the integration of hard real-time systems into the co-simulation [18]. The detailed description of the real-time co-simulation can be found in chapter 2.4.

2.3.1 Definition

Depending on the number of solvers and modeling tools, the divergent categories of simulation are conceptualized (Figure 5). In a classical simulation, a model is solved by one solver (Figure 5(A)). In the interest of simulating complex models, an interdisciplinary model development environment has to be found. In addition, the equation solver must comply with all necessities concerning simulation accuracy and simulation time. It is useful to subdivide it into different subsystems and to calculate it in parallel so that calculation of a model has the potential to be accelerated. The modeling still resembles one simulation environment, but the simulation is distributed (Figure 5(B)). Considering model coupling, individual subsystems are created by dissimilar modeling tools but they are combined and simulated with only one solver (Figure 5(C)). The model parts can thereby be linked as equations or program code or called as a function. The term co-simulation now applies to the utilization of subsystems from several distinct modeling environments or tools and solvers (Figure 5(D)), which solve the subsystems [17],[19].

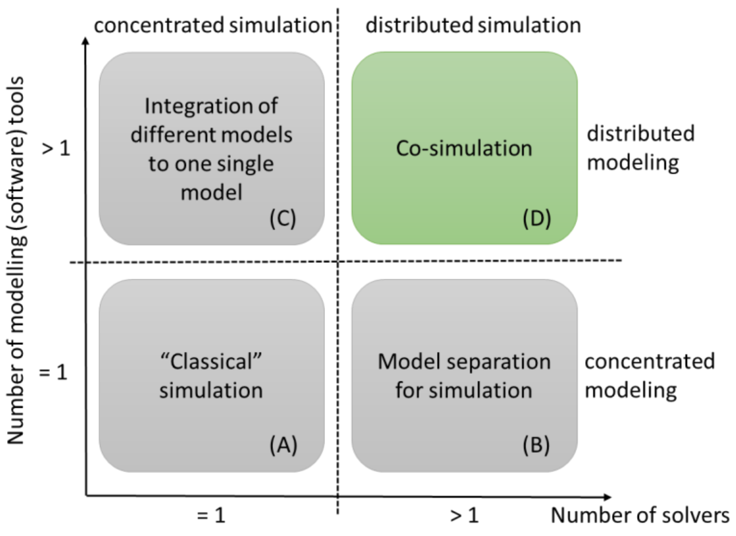


Figure 5: Classification of co-simulation [17]

This requires a program, which is able to offer interfaces to different models and manage the course of a co-simulation.

2.3.2 Challenges for classical co-simulation

The major arduous tasks for a classical co-simulation are [18],[20]:

- Selection of the coupling time instants (Macro step size)
- Definition of the subsystem scheduling
- Choice of an extrapolation method

Co-simulation platforms, which are capable of interfacing with variable simulation tools and transferring data between involved subsystems, can cope with these typical co-simulation challenges.

2.3.3 Choice of macro steps

An important feature of a co-simulation is that involved separate subsystems put their own domain specific solvers and step sizes to use. These step sizes are called micro time steps (δ*t*) (Figure 6). Within a micro time step, the simulation of a subsystem is accomplished once. The data exchange between involved subsystems takes place at pre-defined time steps, which are so-called macro time steps (Δ*T*) or synchronization time. Under the frameworkc of a co-simulation system, subsystems can act in accordance with different macro time steps [19].

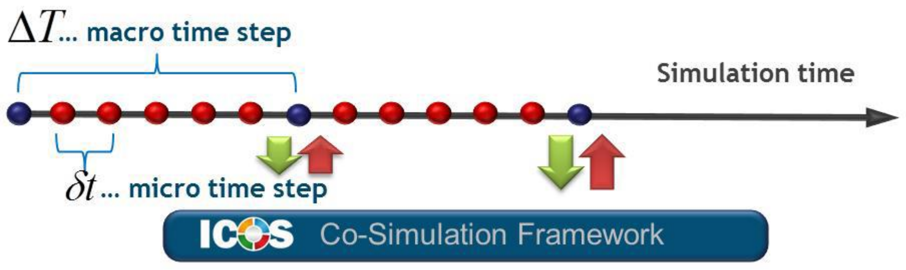

ΔT... macro time step
Simulation time
δt... micro time step
ICOS Co-Simulation Framework

Figure 6: Comparison between micro and macro time steps [19]

It is essential to select a proper macro time step in view of the quality of the overall simulation results because there always exists a tradeoff between the overall simulation period and the simulation results' accuracy. To summarize, more accurate simulation results can be achieved with smaller macro time steps but along with relatively longer simulation duration [18].

2.3.4 Simulation tool scheduling

Generally, there are two possibilities of subsystem scheduling: parallel and sequential. In parallel co-simulation, all input parameters of the involved subsystems have to be extrapolated. This brings a benefit that simulation time can be shortened. However, the simulation results are also degraded by comparatively more extrapolation errors. By contrast, less extrapolation efforts are required for sequential scheduling while it costs longer calculation time. In addition, in some cases the overall system behavior is susceptible to tool scheduling [18].

For example, a controller with a dynamic system is illustrated in Figure 7. At least one extrapolation has to be performed to solve the loop by using the non-iterative approach for the co-simulation (Figure 7(a)). At the first time step, the input variables of the dynamic system have to be extrapolated primarily, subsequently the controller is operated based on the interpolated results of the dynamic system. If the simulation sequence is reversed as demonstrated in Figure 7(b), the input variables of the controller have to be extrapolated at the beginning and interpolated results of it are applied as inputs for the dynamic system [20]. If a parallel scheduling is executed (Figure 7(c)), the input signals belonging to both subsystems are estimated for the first time step. As a result, the simulation time is shortened, but the accuracy and stability of the simulation is likely to be affected.

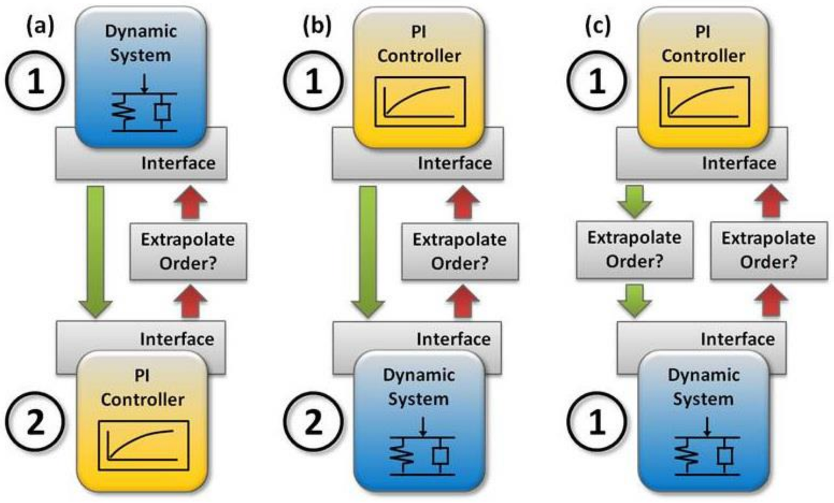


Figure 7: Simulation tool scheduling possibilities [19]

2.3.5 Extrapolation techniques

With the aid of the extrapolation, a signal curve beyond the original observation range is estimated based on past values. The most common methods are the zero-order-hold (ZOH), first-order-hold (FOH) and second-order-hold (SOH), as shown in Figure 8.

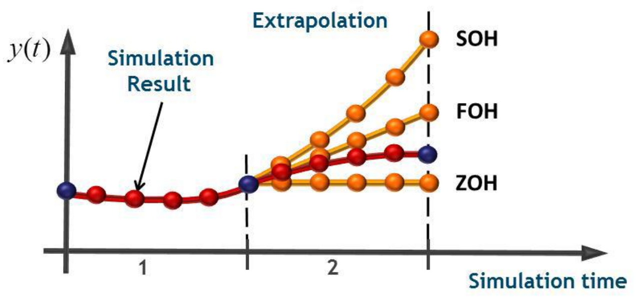


Figure 8: Illustration of different extrapolation techniques [20]

Considering that extrapolation means a prediction of future signals, high risks of coupling errors get introduced [20]. The reduction of coupling errors is pursued since it affects the overall simulation results. Another problem is the introduction of dead-times as the polynomial extrapolation is applied [18]. Due to the dead times, Instability of a closed loop system may occur.

2.3.6 Co-simulation framework

There are two possibilities of co-simulation framework for several subsystems [20]-[21]:

1. Iterative or strong coupling:
   In the iterative coupling (strong coupling), an error threshold is specified and over one macro time step, the subsystem is calculated separately for several times until the introduced error is below the specified threshold. At the first time step, extrapolation of all coupling variables is executed. For each repetition of the macro step, resetting of involved subsystems as well as a reconfiguration of simulation tools must be implemented. Since only a few modeling environments meet these requirements, the iterative method is only applicable to a limited extent.
2. Non-iterative or weak coupling:
   If each subsystem is calculated only once over the time interval of each macro time step, it is called non-iterative or weak coupling. This coupling strategy is usually applied in co-simulation systems. Unavoidably, there exist larger extrapolation errors at each step since only the last quantities are used for extrapolation. The calculation of the involved subsystems can be either in parallel or in sequence.

2.4 Real-time co-simulation

2.4.1 Definition

If at least one hard or firm real-time device or component is integrated in a co-simulation system, it is defined as a real-time co-simulation. The integrated components are possible to be physical sensors and actuators, electronic control units, or simply hard or firm real-time simulations.

Besides difficulties mentioned in a co-simulation before, a real-time co-simulation has to fulfill the requirement of time accuracy. Therefore, synchronization between real-time parts and non-real-time parts presents a huge challenge. In addition, data from physical sensors are commonly corrupted by noises, which handicap the appropriate extrapolation [19].

2.4.2 Real-time Conditions

A system is characterized by real-time when both the timing correctness and the logical correctness of its function is satisfied. The response time of a system is defined as the period between the creation of all input variables and the availability of outputs. The classification of real-time systems is according to the strictness of the real-time restriction on a missed deadline [19],[22] , as shown in Figure 9.

Soft real-time

The quality of a result after the expiry of the deadline distorters along with the delayed response time (Figure 9, left). A user interface can be taken as an example. The user information is determined to be offered as soon as possible, however, short waiting time does not interfere with the usability of the system.

Firm real-time

In firm real-time systems, it is acceptable to miss deadlines occasionally, but the system quality degenerates and system failure might take place in the case of frequent occurrence (depending on the situation). A non-timely result is worthless and useless (Figure 9, center). One application example can be a video conferencing system, in which temporary packet losses can be tolerated.

Hard real-time

Missing the deadline for the response is not acceptable in a hard real-time system since it brings a failure of the system (Figure 9, right). Hard real-time systems are employed when missing a strict deadline possibly leads to catastrophic consequences, such as causing damages to devices or injuries to people. These systems typically interact in a low layer with the embedded systems or physical apparatuses. Instances include the triggering system for an occupant airbag of a vehicle and brake-by-wire systems.

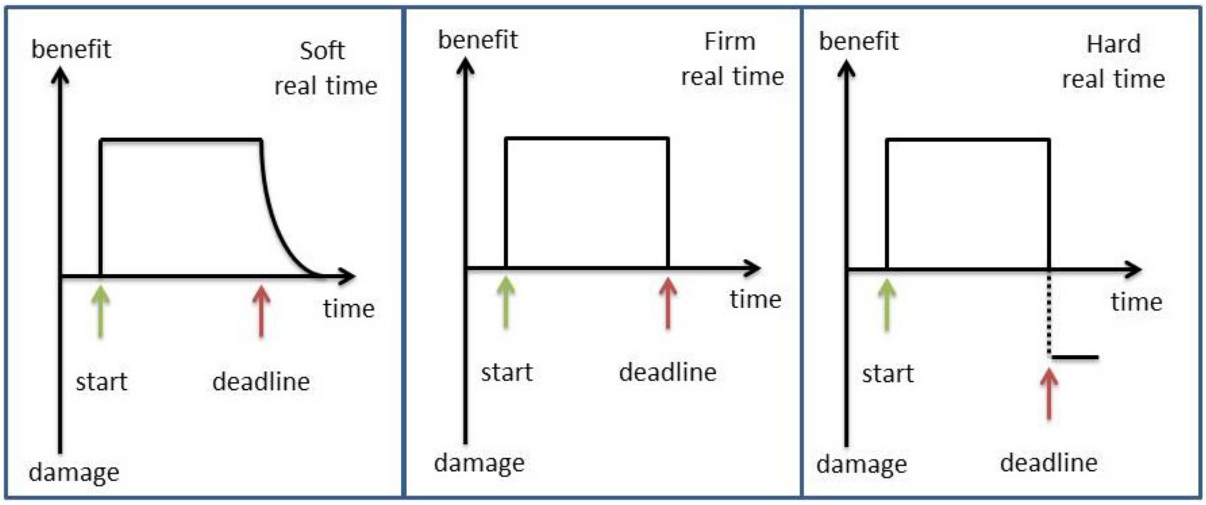


Figure 9: Classification of real-time systems [19]

2.4.3 Classification of the involved Systems

For classification of systems under real-time conditions, two terms are inaugurated [19],[20]:

Online systems

Online systems, whose system time is the same as the wall clock time, are able to fulfill hard real-time conditions. These sort of systems cover hardware-in-the-loop (HiL) systems, physical sensors and actuators, etc.

Offline systems

By contrast, hard real-time requirements do not to have to be satisfied in an offline system. Offline systems can be simulation tools from various disciplines (e.g. MATLAB/Simulink, KULI, AVL Fire, etc.). When an offline system is coupled with an online system, in order to generate appropriate response, the calculation of the online system has to be faster than wall clock time. Otherwise, it is not possible to put them into practice in a real-time co-simulation, such as CFD (computational fluid dynamics) and FEM (finite element method) simulation.

2.4.4 Challenges

Some more challenges for real-time co-simulation are summarized as follows [18]-[20]:

- Extrapolation
  In a real-time co-simulation, the iterative approach is generally infeasible due to the fact that the conditions of hard or firm real-time should be fulfilled. Thus, extrapolation is essential and prevalent polynomial extrapolation approaches are normally applied, such as ZOH, FOH, or SOH. One obstacle of extrapolation is that in a real-time environment like a HiL system, sensor signals tend to be corrupted or degraded by noises.
- Correction of errors
  As discussed before, since extrapolation only attempts to predict the true coupling values, possible errors always arise from this. Therefore, it is important to use approaches, which can reduce errors or correct errors. Specifically, in case that the offline systems do not deliver a response at the appointed/correct time, the missing data from the offline system can be compensated with the aid of the co-simulation framework, so that the simulation is able to continue.
- Handling of dead-times
  Obviously, in a closed loop, transmitting data between two coupled systems takes an amount of time, which is so-called dead-time in the closed-loop system. These dead-times affect the stability of the closed-loop system and infringes hard-real-time conditions as a failure of synchronization might occur. These dead-times can be divided into a receiving dead-time and sending dead-time in a real-time co-simulation system (Figure 10). Approximate compensation of these dead-times is necessary and significant for coupling elements in order that the hard real-time condition can be guaranteed.
- Synchronization
  Generally, offline systems draw on simulation system time instead of wall clock time. In a correct co-simulation environment containing online systems, synchronization of timescales of involved systems must be fulfilled. Only in this case the time correctness of the whole co-simulation can be achieved.
- Choice of macro time step
  To make the real-time systems operate in a normal manner, the macro time step size of coupling has to be less than or equal to the shortest data update time of the real-time subsystems.

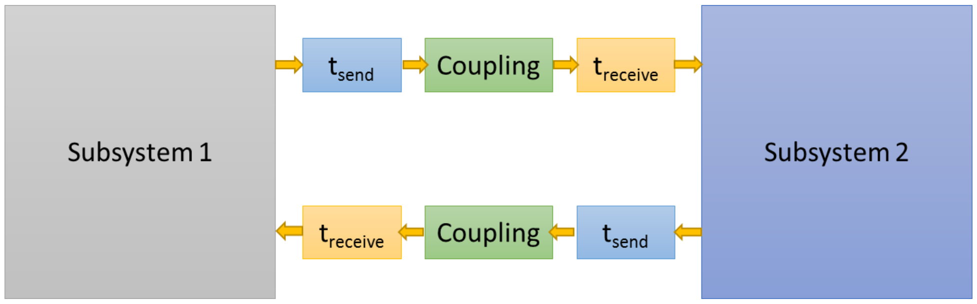


Figure 10: Sending and receiving dead-time due to the data transfer [20]

2.5 Jitter and operating systems

2.5.1 Definition of jitter

Jitter is defined as the timing variations of a series of signal edges from their ideal values. The most common types of jitter are [23]-[24]:

- Cycle to cycle (C2C) jitter: it denotes the difference between the current clock period of a signal and the adjacent one.
- Period jitter: it represents the deviation of a real clock signal transition point from the ideal one over a number of randomly selected cycles.
- Long term jitter: it measures the variation between a clock's output and the ideal position over a number of consecutive clock cycles.
- Time interval error (TIE) jitter: it indicates the deviation of the event of the signal being measured to the reference clock event.

2.5.2 Real-time operating systems

A real-time operating system (RTOS) is a specialized operating system (OS), which is capable of respecting formulated real-time constraints [22]. The RTOS, which is able to schedule tasks at deterministic intervals, is not only being utilized for control applications, but also supports non-control functions like networking and user interfaces. Two kinds of RTOSes exist: the traditional RTOS (including ETS, LynxOS, QNX, Windows CE and VxWorks) and non-real-time OSes enhanced by RTOS extensions (like RTAI and RTL for Linux, and HyperKernel, OnTime and RTX for Windows NT) [25]. Despite the efforts of optimizing RTOSes, jitter is still an issue in these systems due to the hardware effects. The impact of jitter on stepper motor control was analyzed by Frederick M. Proctor and William P. Shackleford [25]. In their work, jitter contributed additional less than 10% torque load of available torque to the motor, but with a compensation algorithm the additional torque was reduced to 1% of available torque. Moreover, it is reasonably presumed that the performance of such experiment with a non-real-time OS is worse.

2.5.3 Analysis of Windows OSes

Almost 90% of operating systems installed in notebooks or desktops are dominated by MS Windows family. Also, the rising number of 64-bit systems is an apparent trend [26]. However, Windows OSes like Windows XP and Windows 7 are not designed to be a hard real-time operating systems (RTOSes) due to the following reasons [26]-[27].

- Limited number of priorities for processes
- Uncertainty of multi-thread operation
- Priority inversion
- Containing critical sections that cannot be interrupted
- Jitter of the timer
- Jitter in the communication with external I/O devices, like a USB link

Fortunately, there exist some commercial real-time extensions of the MS Windows kernels like RTX or RTKernel. They are able to improve the real-time capacity of Windows OSes [26].

2.5.4 Examples of real-time application with a Windows OS

A real-time measurement and control with Windows 7 64-bit OS has been implemented by Krzysztof KOŁEK from AGH University of Science and Technology [26]. In his work, an algorithm that augments the accuracy of the timer was proposed. As an example, Windows 7 64-bit system was used to control a 3D crane model. It can be concluded from the test results that, under his use case and configurations, Window OS is not suitable for controlling hard real-time systems but is still able to fulfil soft real-time tasks with tolerable jitter.

In China, the Windows OS enhanced by RTX (a real-time extension), has been applied to real-time control and measurement in various domains, including satellite attitude and orbit control [28], a HiL test system for laser-guided bombs [29] and a HiL test system for an aircraft [30]. A HiL simulation platform developed by the Computer Science and Technology college of National University of Defense Technology in China is similar to the platform used in this thesis, except that Model.CONNECT only runs in standard Windows OS [27],[31]. The proposed HiL test platform is based on RTX and Simulink, which incorporates hardware interfaces, real-time co-simulation and results analysis. Some new techniques are introduced in the proposed platform, including separation of real-time processes, IPC shared memories with supportive algorithm, real-time I/O driver based on RTX, and I/O interface modules based on Simulink. The structure of the platform is shown in Figure 11.

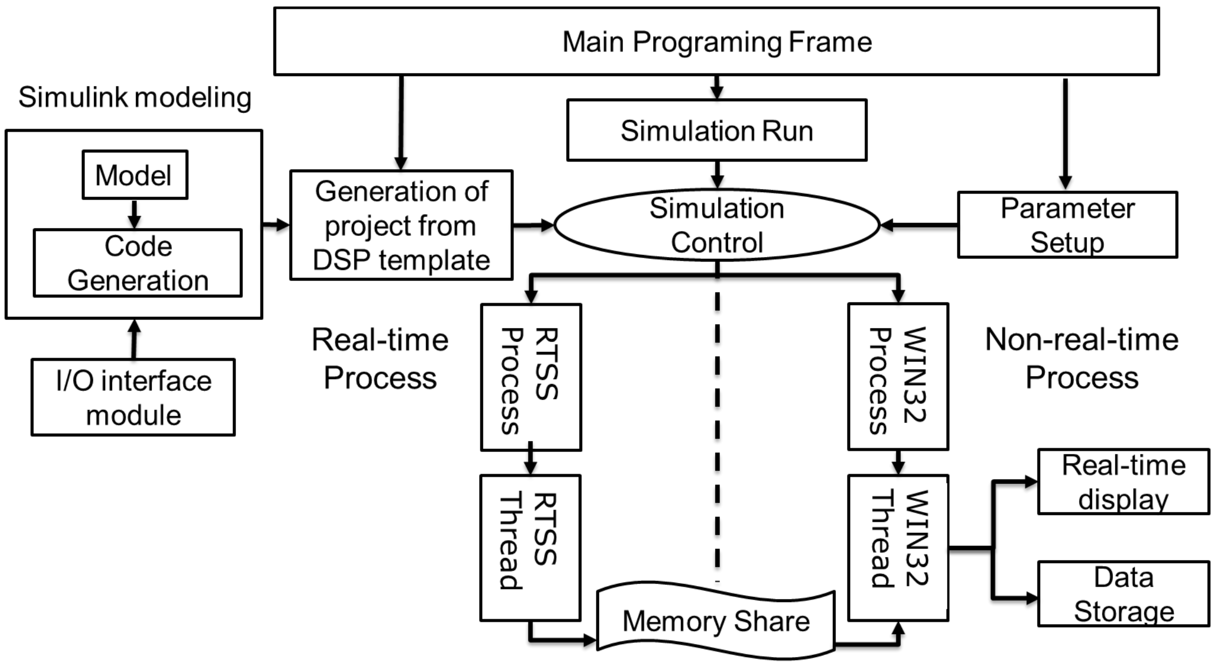


Figure 11: Software architecture of the platform [27]

The prototyped platform has been put into practice and supported HiL tests of a distributed real-time radar system. Some further extensions of the proposed test system are needed [27],[31].

3 Used Programs

3.1 MATLAB

Due to the long-term optimization, the MATLAB platform is an excellent tool to solve technical and scientific problems. By making use of the matrix-based language, it is inherently possible to express computer-assisted mathematics simply. Thanks to implemented graphics functions, data can be visualized and many toolboxes are provided for numerous challenges [32].

32-bit version of MATLAB R2013b and R2010a are used in this thesis. The co-simulation program Model.CONNECT supports all models created by MATLAB version R2009 and later.

3.2 Simulink

Simulink is a block diagram environment, which can be applied in multi-domain simulation and model-based design (MBD). Simulink supports the design and simulation of systems and enables automatic code generation, continuous testing and verification of embedded systems. Furthermore, Simulink comprises a graphic editor, user-defined block libraries, and solvers for modeling and simulating dynamic systems. It is integrated into MATLAB so that MATLAB algorithms can be incorporated in models and simulation results can again be analyzed and further processed in MATLAB [33].

3.2.1 ICOS Interfaces

The data exchange with programs outside Simulink models is implemented by ICOS (Independent Co-Simulation) blocks, which have functions like InPort and OutPort of Simulink (Figure 12). Via these interfaces, Simulink models are ready to exchange signals with other simulation programs or hardware in the framework of the co-simulation platform Model.CONNECT. If these blocks, which are available in the Simulink Library after the Model.CONNECT installation, are inserted into a Simulink model, the inputs and outputs are visible when the model is linked to the co-simulation program.

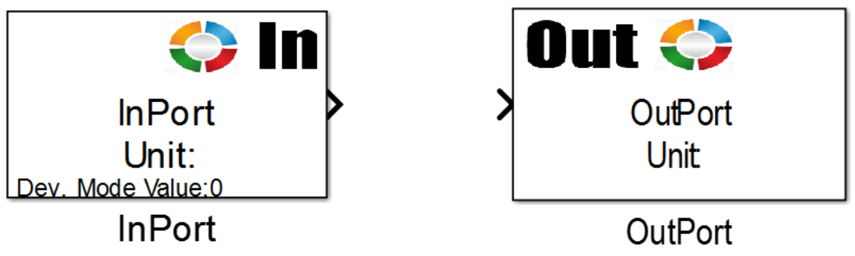


Figure 12: ICOS Blocks in Simulink as interfaces to Model.CONNECT

It should also be noted that in this thesis only signals with the double data type are transmitted, which means that the data types of the signals shall be converted before transferring. Other setting parameters of these blocks are limited to the signal name and the dimension of the signal [34].

3.2.2 Vectors

In Chapter 6 of this thesis, the communication between online and offline systems is focused, because it affects the real-time capacity of the overall simulation. One overriding factor, which leads to deterioration of the real-time performance, is the synchronization of a large number of data. Vectors shall be used for the purpose of transferring the data more efficiently.

Using Mux and Demux blocks, several signals can be combined into one vector and one vector can be outputted as separate single signals again. The advantage of this idea (Figure 13) lies in the reduction of the computation time. For each ICOS block an S function is executed in Simulink, which requires extra time in addition to the actual data transmission. In case of 400 interfacing signals, this additional computing cost is drastically reduced by the utilization of vectors.

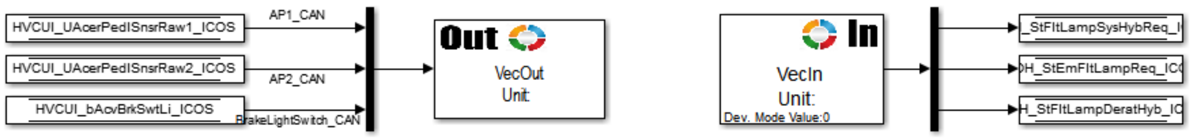


Figure 13: Data transfer with vectors

3.3 AVLab

AVLab is a proprietary development by AVL List GmbH and is the successor of the internal development program PoET (Powertrain software Engineering Tool). It is used for the development of control software. AVLab v2.9.8 was used within this master thesis.

The main advantages of this program are [35]:

- Several tools, which support the development stages from modeling and testing to code generation, are integrated into AVLab. Therefore, efficiency and quality of the functional development can be enhanced.
- Signal properties in Simulink can be synchronized with ADD using a synchronization tool.
- The Integrity program used for documentation and version management is also supported by AVLab.
- AVL Concerto is used for the visualization and verification of signal sequences and can be used directly in AVLab.
- Tests like Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) can be easily implemented with AVLab.
- The stimuli generator allows people to load, create, edit and store various data, and to use them later as excitation signals for a Simulink model or for the ETAS LABCAR operator.
- For code generation, a toolbar is provided, which provides code-compliant code generation for TargetLink code and embedded coders.
- Simulink model development is based on a uniform template, which utilizes the AVL Powertrain Software Library component structure. It comprises an operating system that allows realistic simulations as a pure simulation in Simulink

3.3.1 Simulink Structure

The structure specified by AVLab is shown in Figure 14 and a brief introduction to each block is given in the following part [36].

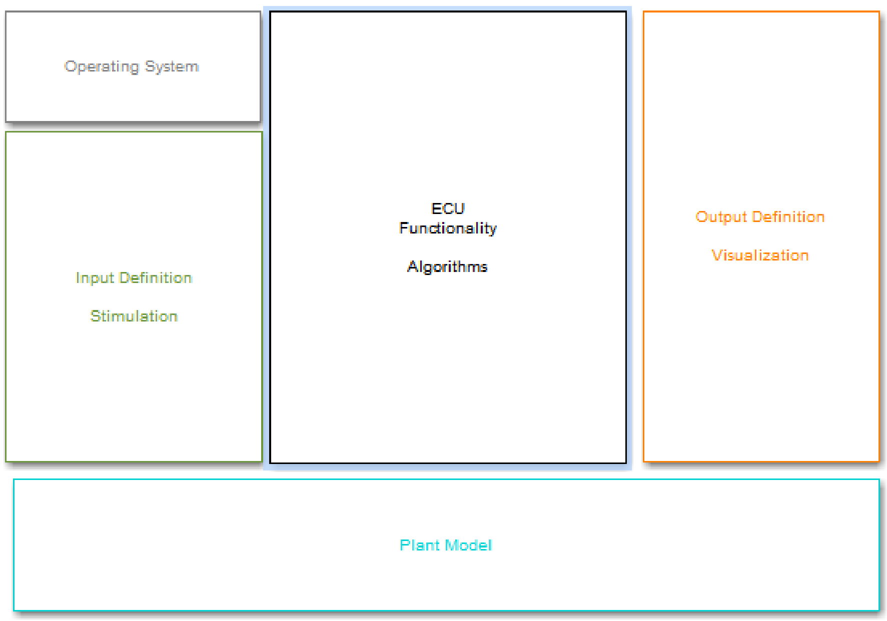


Figure 14: AVLab standard structure in Simulink

On the upper left is the operating system of the control unit, which is one of the most significant features of the model template. It makes it possible that simulation of functions with a scheduling is similar to the scheduling in a real ECU. Below this, in the Input Definition block, predefined DataStoreWrite blocks are used to feed signals generated by AVLab to the respective test case. In the output definition block, scopes or displays can be inserted, but this function is not used in this thesis. The module in the middle, ECU Functionality Algorithms, consists of many layers. In the first layer there is a subsystem, which is addressed by the operating system with the cycle time and is therefore executed and calculated once every 10 ms (Figure 15). It is a function-call subsystem, which is called cyclically as a function.

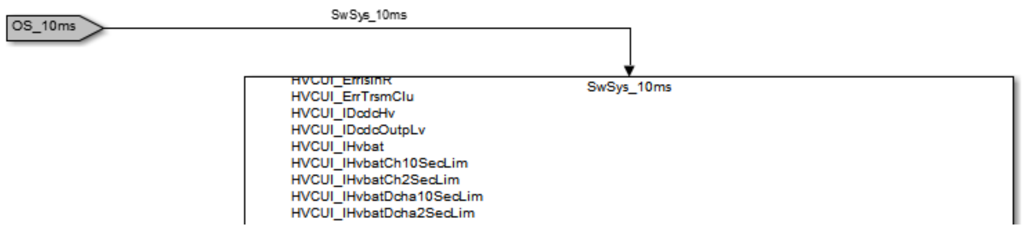


Figure 15: The first layer of ECU Functionality Algorithms

Another layer underneath is SwSys_Test block created by TargetLink. This block is used to generate executable programming codes. For easier handling of the inputs and outputs, all signals are bundled into buses (Figure 16).

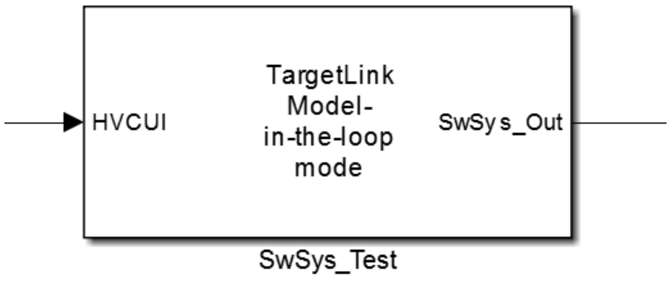


Figure 16: SwSys_Test block

Inside the SwSys_Test block, there is a Simulink model of the control software, which is inserted as a referenced model (Figure 17).

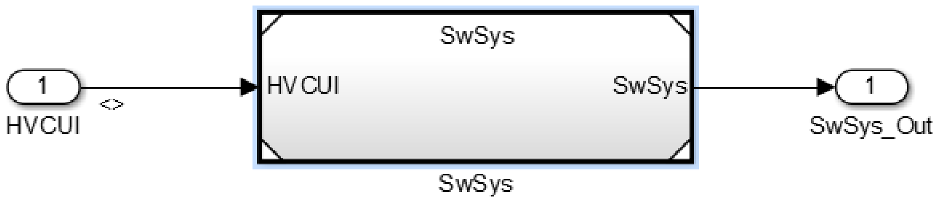


Figure 17: Referenced model for control software

3.3.2 Simulation process

The operation of AVLab during the simulation has several steps. If a new AVLab project is created, a Simulink model with the already described basic structure is created and stored under the name of XXX_Test. This model is used for all further developments and should not be renamed, otherwise the start from AVLab is no longer possible. If a simulation is started by AVLab, AVLab creates a copy of this model, renames it into XXX_miltest. And some other changes take place in XXX_miltest: A stimuli (Signal Generator) is inserted, which is used for generation of inputs from the test case. A block called initLoggedSignals is created, which is used for the initialization of logged signals. Additionally, the callback functions are written into programming code and the simulation stop time as well as SimTimeConst is changed too.

Meanwhile, the original model XXX_Test is not modified during the simulation. This means that the manual change of the model is only valid in the original model when using AVLab to start the simulation. Furthermore, if the simulation is started by AVLab, only the copy of the original model is applied in the simulation.

3.4 Model.CONNECT

Model.CONNECT is a co-simulation platform, which was developed by AVL List GmbH. It offers numerous interfaces to integrate models from multiple simulation environments. With the assistance of Model.CONNECT, it is possible to combine simulation data from different technical disciplines and domains with each other. Also, communication with real-time applications such as an ECU can be easily established and implemented.

The abbreviation MC is used for the following mentions of Model.CONNECT.

MC was firstly created in 2015 and it has been continuously developed since then. In this work, many versions of MC were used in this thesis and R2017f Build 091 was lastly applied.

3.4.1 Setup of the simulation environment

After creation of a new project, the simulation models are integrated into the topology window by the means of corresponding blocks. The interfaces of each simulation model then appear in MC as ports, which can be easily connected with those of other models.

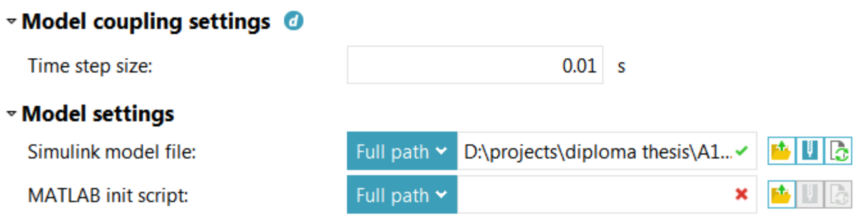
For this work, only the interfaces to MATLAB and CAN buses were needed, the latter in this work is produced by RealTime wrappers (Figure 18). An alternative to RealTime wrapper is the CAN Wrapper, which might have edges over the RealTime wrapper when a large number of CAN signals are confronted and the *.dbc file is available.

Figure 18: MATLAB, RealTime and CAN wrapper

3.4.2 Integration of Simulink models

The integration of a MATLAB/Simulink interface requires a specific parameterization based on the project. The most important steps are explained below.

The first move is to define the time step, in which the inputs and outputs of the model are updated, as shown in Figure 19. The time step size corresponds to the macro step size and must not be less than the simulation step size in the corresponding Simulink model. If the Simulink model uses variable time step, the macro step size or the time step size in MATLAB wrapper must not be less than the maximum value of the variable step size of the Simulink model.



Model coupling settings

Time step size: 0.01 s

Model settings

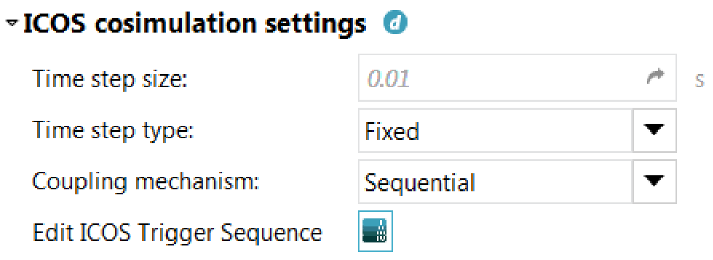
Simulink model file: Full path D:\projects\diploma thesis\A1...

MATLAB init script: Full path

Figure 19: MATLAB wrapper parameters

The path to the model is specified in the Simulink model file path (Figure 19). Simulink models with the file extensions *.mdl and *.slx are accepted.

It is possible to run the simulation, which consists of more than two models sequentially or in parallel. The selection of sequential or parallel co-simulation can be found in ICOS co-simulation settings (Figure 20) and the settings of the calculation sequence can also be defined by ICOS trigger sequence number (Figure 21).



ICOS cosimulation settings

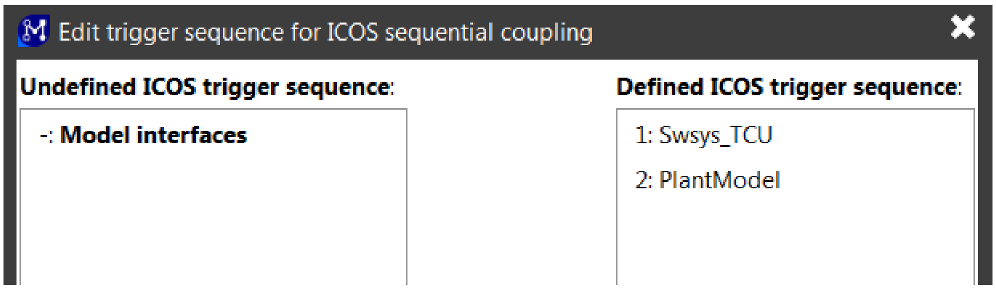
Time step size: 0.01 s

Time step type: Fixed

Coupling mechanism: Sequential

Edit ICOS Trigger Sequence

Figure 20: ICOS co-simulation settings


Edit trigger sequence for ICOS sequential coupling

Undefined ICOS trigger sequence:

-: **Model interfaces**

Defined ICOS trigger sequence:

1: Swsys_TCU

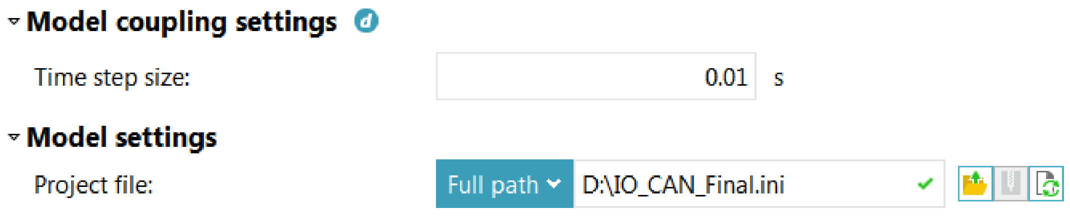
2: PlantModel

Figure 21: Definition of ICOS trigger sequence

3.4.3 Integration of a CAN bus using the RealTime wrapper

The ICOS RealTime wrapper is responsible for the communication with the real hardware via CAN buses. PCAN-USB, PCAN-PCI devices or Vector CAN adapters are supported by this block, therefore they can be used in this work. In addition, PEAK-System's devices are also used to simulate the sensors and actuators of the control unit, and these devices are more cost-effective than their competitors' products [34].

For the RealTime wrapper, the time step size setting is similar to the setting of MATLAB wrapper before, but the model is referenced to a project file (Figure 22). The property of the CAN bus is specified in this * .ini file. The exact structure of this file is given in the MC manual [37].


Model coupling settings

Time step size: 0.01 s

Model settings

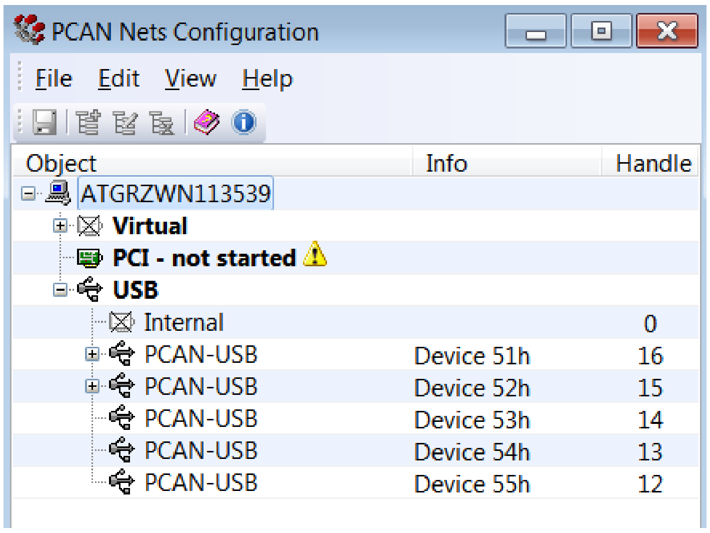
Project file: Full path D:\IO_CAN_Final.ini

Figure 22: RealTime wrapper settings

Firstly, an identified port must be created for each signal, which is then displayed in MC on the RealTime block, and the extrapolation type must also be specified. Due to the fact that detailed information for each signal is deficient, the zero-order-hold option, which is defined after the signal name in an *.ini file, is utilized for all signals. In CAN settings, the number of the handle of a device must be entered in the *.ini file for specifying CAN channel, as well as the CAN baud rate. The correspondence between the CAN baud rates and numeric code can be found in a table of MC user manual [37]. Some other definitions are also needed for each signal, including the ID of a message, start bit, end bit, factor and offset.

In the *.ini file, the application of a *.dbc file is possible, which reduces the efforts of entering the information of signals manually. The path of the *.dbc file must be specified in the *.ini file as well.

If several PCAN-USB adapters are used, a unique identification number has to be assigned to each of them. Identification number is set by the program PCAN-View and ranges from 0x51 to 0x58. If PCAN-USB adapters are connected with a computer, each of them is automatically assigned to a handle. With the PEAK program PCAN nets Configuration (Figure 23), all devices for CAN bus communication are listed with their handles.



Object	Info	Handle
ATGRZWN113539		
Virtual		
PCI - not started		
USB		
Internal		0
PCAN-USB	Device 51h	16
PCAN-USB	Device 52h	15
PCAN-USB	Device 53h	14
PCAN-USB	Device 54h	13
PCAN-USB	Device 55h	12

Figure 23: PCAN nets Configuration

The assignment of the handles to the devices follows the chronological order of the connection to the computer. So the first PCAN-USB connected to the computer is assigned to the handle 16 and therefore it has also to be assigned to the CAN channel 16 in the * .ini file. The next connected device is assigned to the handle 15, the next 14, etc.

To summarize, the PCAN-USB must be connected with the computer in a specific sequence, to ensure that the number of the handle of a device is as same as the CAN channel defined in the *.ini file.

3.4.4 ICOS Embedded Mode

It is achievable to start a co-simulation out of MC. In this work, the co-simulation can be started by MATLAB out of MC.

There are some details, which need to be taken care of. Firstly, it is not good to put the MC project under the path of the MATLAB workspace.

And it is suggested to delete the blank in the name of the block in MC, which is to be the master element. For example, if the master element's name is MATLAB 1, then a *.m file called MATLAB_1 is created under the model path, which might bring some confusion.

Finally, it should be ensured that the automatically created *.m file used for ICOS embedded mode do not have the same name as any other models or files under the path of MATLAB. Otherwise the created *.m file might not be executed when using ICOS embedded mode.

3.4.5 Importing and exporting of connections

In MC, there is a fast way to acquire the connections between blocks by exporting and importing connections (Figure 24), whose format is *.CSV file. A newly created project is able to reuse the old connections in previous projects since the old connections can be exported from the previous project and imported into the new one. Also, it is possible to create a *.CSV file by the means of MATLAB programming, which is described in chapter 6.5.1.

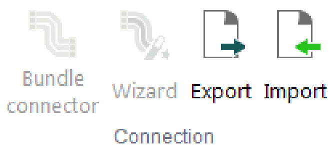
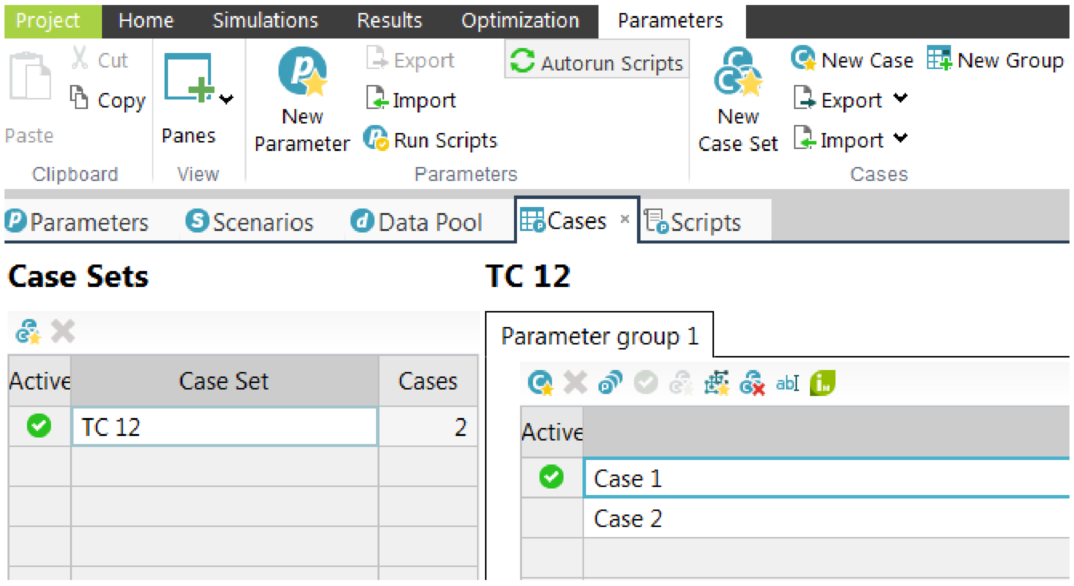


Figure 24: Importing and exporting of connections in MC

3.4.6 Setting of Cases and Parameters

Under the tap of Parameters in MC, more case sets and cases can be created and renamed (Figure 25). If there is a tick in front of the case, it means that this case is active currently and the modifications of the project will be applied to the currently active case.

Utilization of this setting is beneficial to management of multi-case simulation and organization of the results. Moreover, some parameters (e.g. Model path) can be created to reduce the repeatable manual work.



Active	Case Set	Cases
✓	TC 12	2

Active	
✓	Case 1
	Case 2

Figure 25: Setting of Cases and Parameters

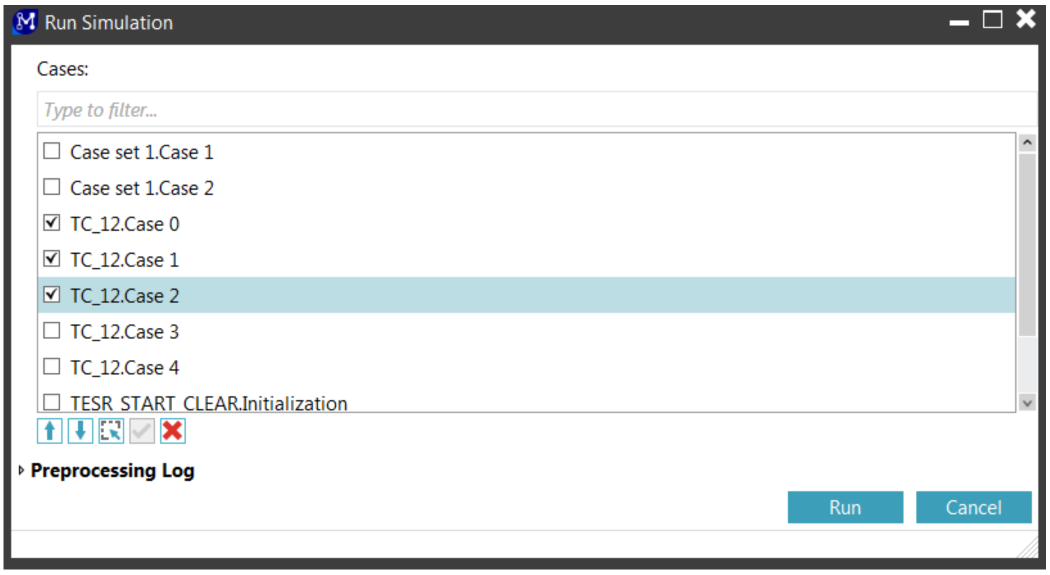
3.4.7 Job Management System

The Job Management System (JMS) offers multiple features and standard GUI interfaces for various AVL products, which support job submission, control and monitoring. Aided by JMS, different host configurations beginning with a simple localhost, the submission to remote hosts or even a queuing system becomes reliable [38].

There are four kinds of case simulation types:

- Single case simulation
- Multi-case simulation using Run Simulation
- Multi-case simulation on a remote host
- Multi-case simulation from the command line

In the event of a multi-case simulation using Run Simulation, if the Job Overview window is on the top, after clicking Run, cases to be implemented can be selected in a window (Figure 26). The selected cases are able to run parallel or sequentially.


Run Simulation

Cases:

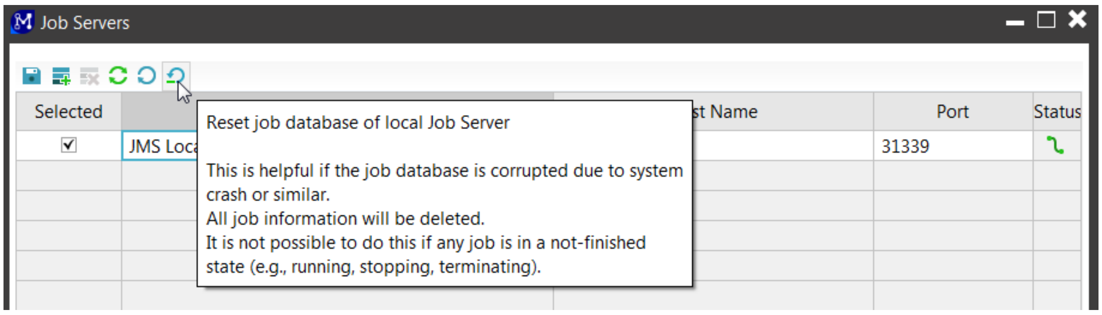
Type to filter...

☐ Case set 1.Case 1
☐ Case set 1.Case 2
☑ TC_12.Case 0
☑ TC_12.Case 1
☑ TC_12.Case 2
☐ TC_12.Case 3
☐ TC_12.Case 4
☐ TESR_START_CLEAR.Initialization

Figure 26: Multi-case simulation using Run Simulation

After a simulation, the submission, which has run, can be reused simply by clicking Rerun.

If the simulation cannot be started again, or some unexpected errors or crashes take place, then resetting the job server might be helpful (Figure 27). After this action, all old jobs or submissions will be cleared.


Job Servers

Selected		st Name	Port	Status
☑	JMS Loca		31339	

Reset job database of local Job Server

This is helpful if the job database is corrupted due to system crash or similar.
All job information will be deleted.
It is not possible to do this if any job is in a not-finished state (e.g., running, stopping, terminating).

Figure 27: Reset job database of local Job Server

3.4.8 Automatic testing using manual job submissions

By applying JMS, the automatic testing of a series of test cases is attainable. The detailed measures are described as follows [38]:

Prerequisite of the automatic testing is to set up manual job submissions, which allows all required input files to be generated without starting the simulation immediately.

There are two options available [38] :

- Exchange of state and message information is handled by a job server and monitoring from the standard GUI is also enabled.
- The state information is file based without using a job server (e.g., job state file in the case directory).

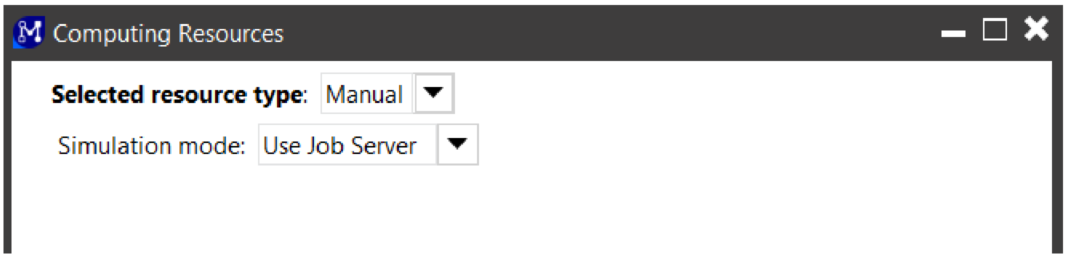
Take the option using job server as an example, the following steps should be performed:

1. In the Computing Resources | Manual dialog, Manual and Use Job Server should be selected (

Figure 28).

2. The simulation job is proceeded by clicking Run. After some time, the simulation job is in the “Ready” status and a new submission is created in the *.job file in the directory of the project.

3. The jms_run should be called by Command Prompt (cmd.exe) in order to start the simulation.


Computing Resources

Selected resource type: Manual

Simulation mode: Use Job Server

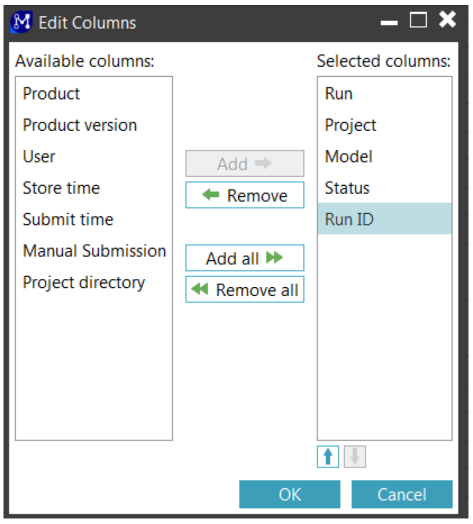
Figure 28: Computing Resources parameters

The jms_run.exe can be found in the installation directory of MC. So in the cmd, the first step is to change the working directory. After this, the jms_run.exe is able to activate MC model. There are two possible usage grammar of jms_run in the cmd:

jms_run [run_id] [model_directory]

jms_run run_id model_directory [cases|tasks] [command]

The Run ID appears after adding the Run ID to the column pane in configuring column window. (Figure 29).


Edit Columns

Available columns:

- Product
- Product version
- User
- Store time
- Submit time
- Manual Submission
- Project directory

Selected columns:

- Run
- Project
- Model
- Status
- Run ID

Figure 29: Edit columns

3.5 PEAK programs

For the purpose of simulation of sensors and actuators for the control unit, devices from PEAK-System were used, which provide analog as well as digital input and output signals. By the means of combining several different devices called MicroMod, the entire inputs and outputs of the control unit could be covered.

In this work, Two MicroMods with analog I/O and one with digital I/O were used. They were connected with MC via CAN buses. With the aid of the software PCAN-MicroMod Configuration, the conversion from analog to digital data or from digital to analog data can be conveniently handled.

3.6 CANape

CANape is a tool related to all tasks of ECU optimization, which is developed by Vector Informatik GmbH. The application field of CANape involves measurement, calibration and diagnostics. CANape offers an easy and powerful solution to diversified tasks, from measurement and calibration of internal ECU parameters, analysis of measurement data, to access to diagnostic data and services. Another advantage of CANape is that it can be implemented for the desktop, for the test bench or in the vehicle [39].

3.6.1 CANape Windows

CANape provides many different windows, which can assist the user in debugging, calibration, obtaining data as well as other information during the measurement and inspecting the status of the device.

One example is the graphic window, which is used to display the variation of measured signals over time (Figure 30).

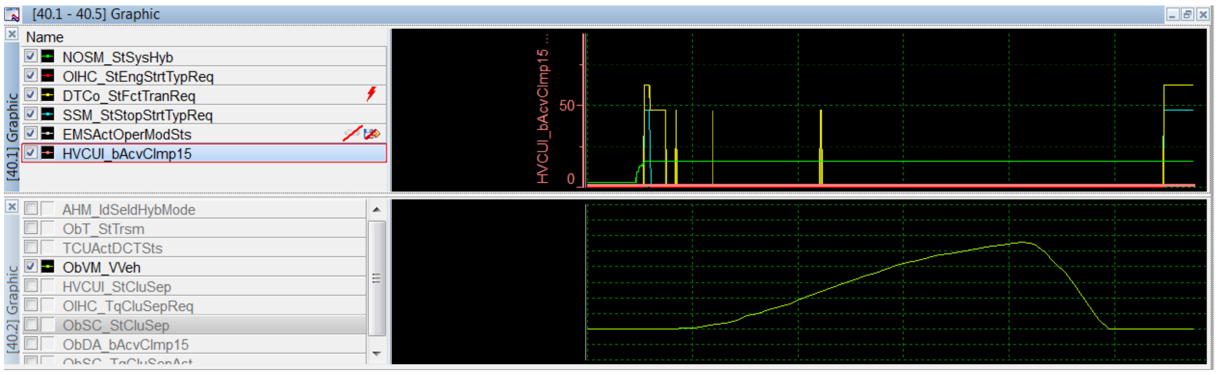


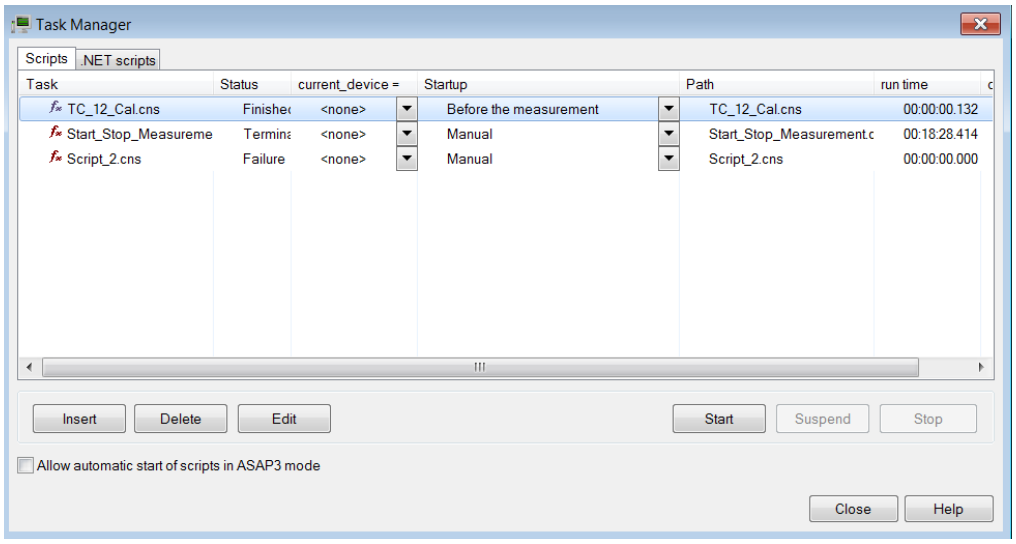
Figure 30: One example of the graphic window

3.6.2 Functions and scripts

In CANape many cross-device functions can be defined. The definition of an arithmetic or algebraic formula in the function is available, in which variables act as placeholders for parameters or real signals. It is also achievable to write a program in a C-like programming language. The function is started by triggering event and executed synchronously during a measurement [40].

Another possibility to fulfill some functions is the utilization of a script. A script can be started independently from a running measurement, and it can be called asynchronously. In this thesis, the script is selected, because during the period before and after the measurement, some functions also need to be implemented.

The definition of a function or a script can be accomplished in Functions Editor. Moreover, in Task Manger it is convenient to run scripts either triggered by certain conditions or simply manually (Figure 31).



Task Manager

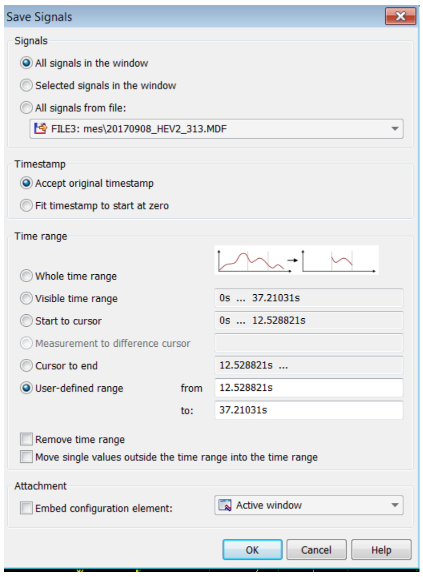
Task	Status	current_device =	Startup	Path	run time
TC_12_Cal.cns	Finishe	<none>	Before the measurement	TC_12_Cal.cns	00:00:00.132
Start_Stop_Measureme	Termin	<none>	Manual	Start_Stop_Measurement.c	00:18:28.414
Script_2.cns	Failure	<none>	Manual	Script_2.cns	00:00:00.000

[ ] Allow automatic start of scripts in ASAP3 mode

Figure 31: Task Manger

3.6.3 Saving signals

After a measurement, CANape supports saving the useful time range of a measurement and neglecting the invaluable time range. In the window of saving signal, the time range that is useful can be easily selected and saved as another *.MDF file (Figure 32).



Save Signals

Signals
- [✓ All signals in the window]
- Selected signals in the window
- All signals from file: FILE3: mes\20170908_HEV2_313.MDF

Timestamp
- [✓ Accept original timestamp]
- Fit timestamp to start at zero

Time range
- Whole time range
- Visible time range: 0s ... 37.21031s
- Start to cursor: 0s ... 12.528821s
- Measurement to difference cursor
- Cursor to end: 12.528821s ...
- [✓ User-defined range] from 12.528821s to: 37.21031s

[ ] Remove time range
[ ] Move single values outside the time range into the time range

Attachment
[ ] Embed configuration element: Active window

Figure 32: Saving useful time range

3.7 Concerto

AVL Concerto is a data processing platform, which offers multiple sophisticated reporting functions for visualization and analysis of different measured or simulated data. It has open interfaces so that it empowers the data correlation tool for different application fields, like simulation, automation or measurement systems [41].

In this thesis, the benefit of the application of Concerto is that the measurement file (*.MDF) from CANape can be loaded into Concerto and thereby some sophisticated pictures can be effortlessly acquired. Moreover, with some additional scripts, the automatic evaluation of the test results in Concerto is achievable.

4 TCU virtual test

4.1 Introduction to the Transmission Control Unit and its original test environment

The Transmission Control Unit (TCU) software is developed in Simulink based on MATLAB 2010a and applied both in a conventional drivetrain and hybrid drivetrain. It consists of numerous subsystems, which fulfil various functions, like the actuation of changing of gears, opening and closing the clutch or the respond for the driver's torque demand.

Meanwhile, the plant model, which reflects the vehicle with all its features and control devices, also locates in the same Simulink model. The core of the plant model is an AVL CRUISE model, in which whole vehicles can be simulated by assembling individual vehicle components. These include, for example, the characteristics of the engine and the electric motor, the air resistance, the braking performance or the characteristics of the transmission. All control units, such as a simplified HCU (Hybrid Control Unit) or BMS (Battery Management System), are executed as simple modules in the plant model.

The PoET_tester supplies the inputs to the plant model and is able to simulate the demand from a driver. However, because the PoET (Powertrain Engineering Tool) is not supported any more, it is not utilized in the TCU virtual co-simulation.

The overview of the TCU, the plant model as well as PoET can be found in Figure 33.

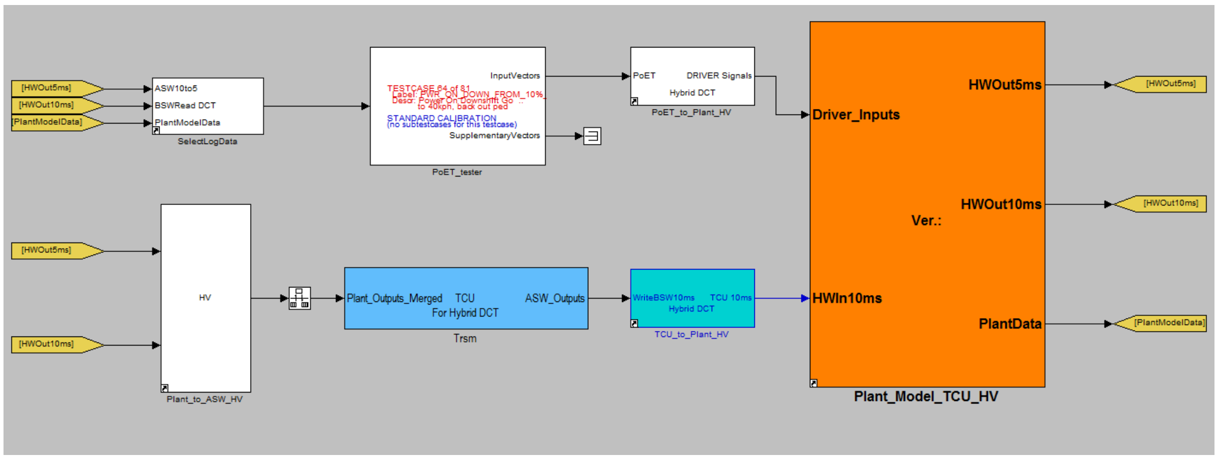


Figure 33: Original test environment of TCU

4.2 Test arrangement of the TCU virtual co-simulation

4.2.1 Test topology

The original model-in-the-loop (MiL) test of TCU was only implemented in Simulink, which can expressed by the Figure 34.

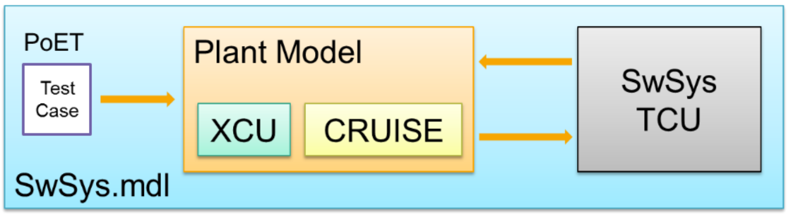


Figure 34: Topology of original MiL test environment for TCU

However, in this work, the simulation is extended in next steps by the co-simulation program MC. The exchange of the data between the TCU and the plant model no longer takes place in one Simulink model of one MATLAB workspace, but in two MATLAB workspaces with data transmitted by MC. This test arrangement is used to validate the concept of co-simulation and compare the performance of running time.

The TCU model is taken from the original test environment and it communicates with the plant model exclusively via MC (Figure 35**)**.

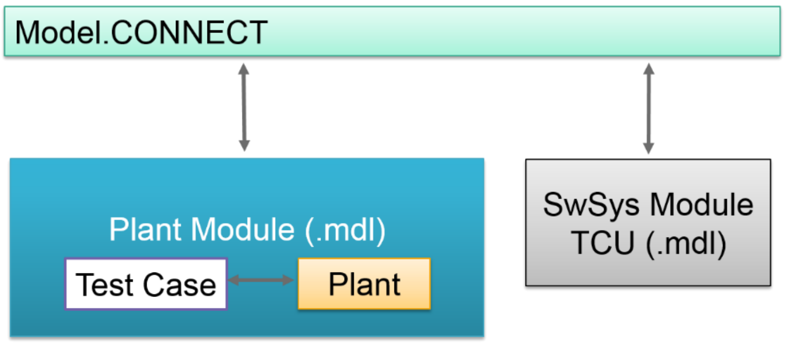


Figure 35: Topology of TCU virtual test

4.2.2 Workflow

The workflow of the TCU virtual test is descried as follows (Figure 36). The first stage is to separate TCU and plant into two different Simulink models while ensuring that they can run independently without errors. Following this, in each single Simulink model the ICOS interfaces should be created so that the interface can be identified by MC. Sequentially, The MC project is correctly created and set up. If the MC project can run without any error, eventually the test result should be validated by comparing co-simulation results with the original test results.

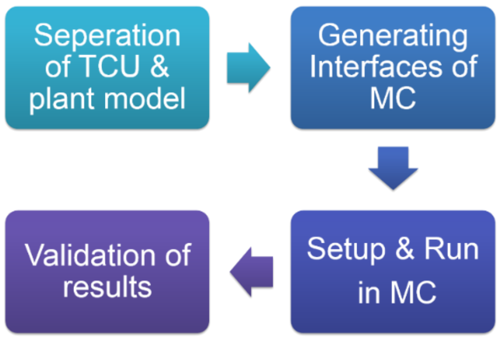


Figure 36: Workflow of TCU virtual test

4.3 Changes to the Simulink models

In order to create interfaces between MC and Simulink models, the latter have to be modified and extended.

4.3.1 ICOS interfaces

For the purpose of increasing data transmission efficiency and decreasing the calculation time of each model, the ICOS interfaces transferring vectors are utilized.

In terms of TCU model, the input signals (113 signals totally) are divided into two groups, according to the sample time of each signal. Each group of signals are received by an ICOS interface with corresponding dimensions (dimensions should be equal to the number of signals transmitted). Since ICOS interfaces employ the data type of double, before and after ICOS blocks, conversion of data type is necessary, as well as renaming of the signals. Following this, the input signals for TCU model are combined into buses again and transferred to the TCU Software. Meanwhile, only one OutPort of ICOS interface is applied, due to the fact that all output signals (35 signals totally) have the same sample time. Before the ICOS OutPort block, the bus is converted into vector and the data type of signals is converted as well. The simplified ICOS interfaces can be seen in Figure 37.

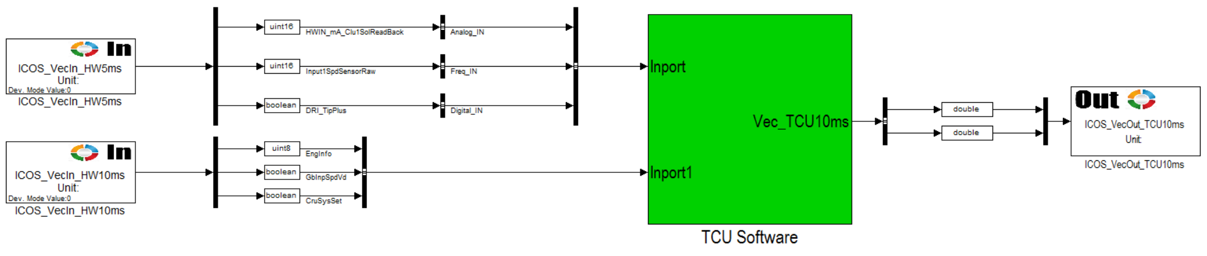


Figure 37: Simplified ICOS interfaces of TCU model

Similar changes are also implemented in plant model, which can be seen in Figure 38.

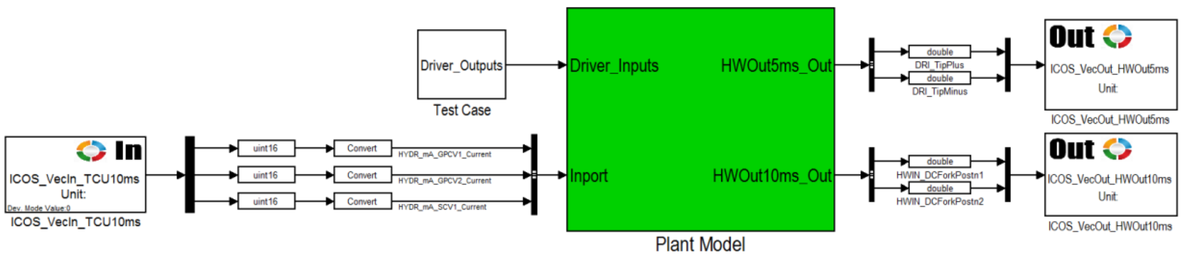


Figure 38: Simplified ICOS interfaces of plant model

4.3.2 Changes to simulation step

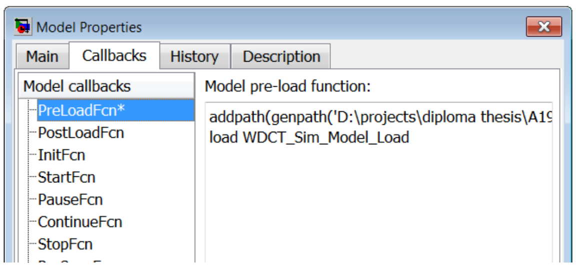
For the original software, in the solver options, the time step type is variable and the max step size is 0.005s. In MC, the macro step of a component must be not less than its micro step (here the step size of the Simulink model). The time step of Simulink simulation is changed into fixed step with 0.001s in both plant model and TCU model, so that the calculation step in models is uniform and a smaller macro step (0.001s) can be realized in MC. This can enhance the co-simulation accuracy theoretically.

Additionally, some rate transition blocks are inserted in order that there is no simulation error.

4.3.3 Additional setting of Callbacks

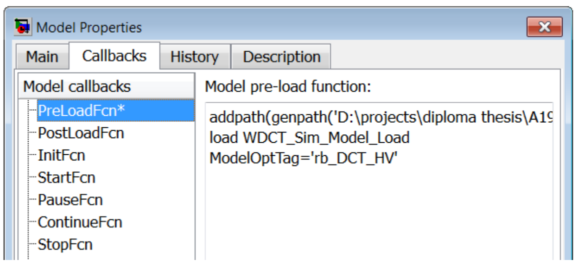
In the primary test environment, some configurations and parameters were loaded before a simulation started. In order to run the co-simulation without errors, pre-loading of necessary parameters for the models has to be carried out.

In the TCU model, as shown in Figure 39, firstly the path of all library blocks is added, and then all parameters in the workspace of the original software are loaded (Figure 39(a)).



```
addpath(genpath('D:\projects\diploma thesis\A19
load WDCT_Sim_Model_Load
```

(a) Callbacks of TCU model



```
addpath(genpath('D:\projects\diploma thesis\A19
load WDCT_Sim_Model_Load
ModelOptTag='rb_DCT_HV'
```

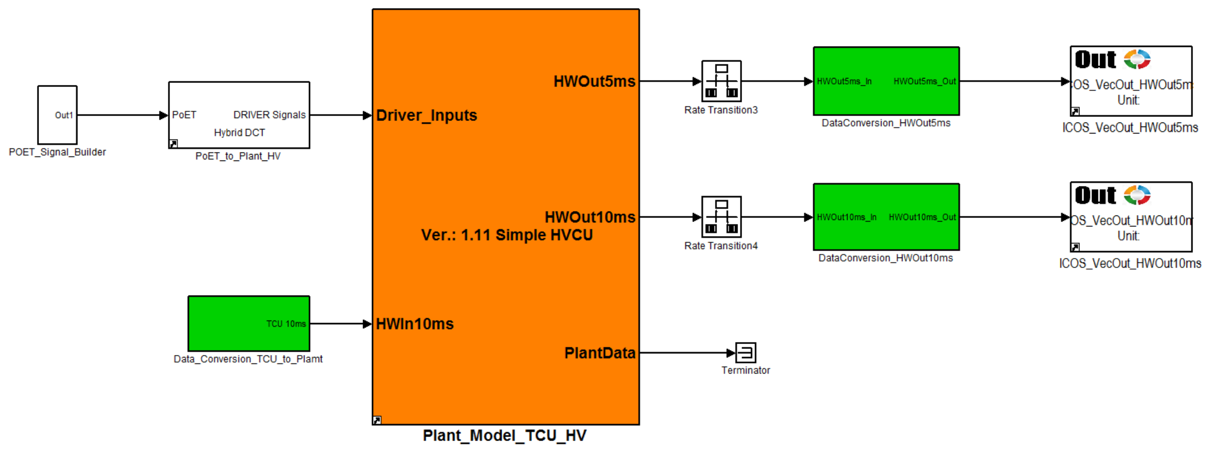
(b) Callbacks of plant model

Figure 39: Callbacks of TCU model and plant model

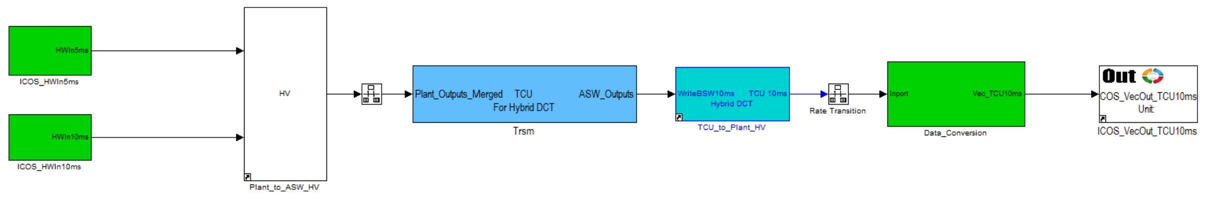
Considering the plant model, similarly, the same path is added and parameters are loaded. In addition, a variable, ModelOptTag is set to a string "rb_DCT_HV" (Figure 39(b)). This is because the original test environment contains two types of plant model, a conventional car or a hybrid car. In this thesis, only the hybrid car is applied and corresponding calibration is loaded into the MATLAB workspace.

4.3.4 Final TCU model and plant model

Eventually, the accomplished separate TCU model and plant model can be seen in Figure 40.



(a) Final plant model



(b) Final TCU model

Figure 40: Final TCU model and plant model

4.4 Model.CONNECT setup

By linking the TCU software and Plant model, data between two Simulink blocks now can be exchanged via three vectors (Figure 41). The step size (macro time step) of both blocks is equal to the cycle time of two Simulink models, which is 1 ms.

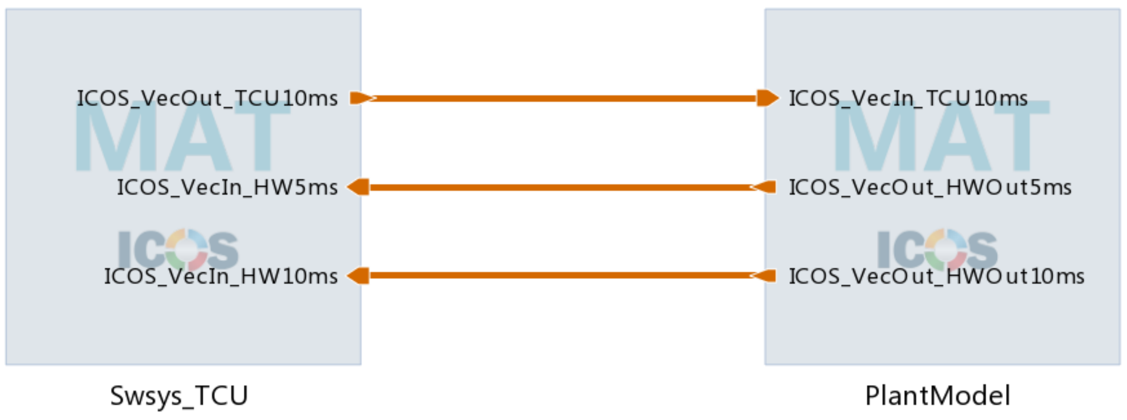


Figure 41: Model.CONNECT topology of TCU virtual test

4.5 Test case

In the original test environment, a tool called POET is used for the simulation of driver inputs. In this thesis, with the aim of avoiding simulation troubles, the POET is not utilized but the test case inside POET is saved as a signal builder, and applied in the further work.

The test case, LAUNCH_PWR_ON_10%_IN_R_GEAR_&_PWR_ON_50%_IN_D is used. In this test case, the simulated car firstly launches in gear R and then stops and finally drives in gear D, which covers most of the operation conditions that the car meets. This is why this test case is opted for.

The important signals in the test case can be seen in Figure 42.

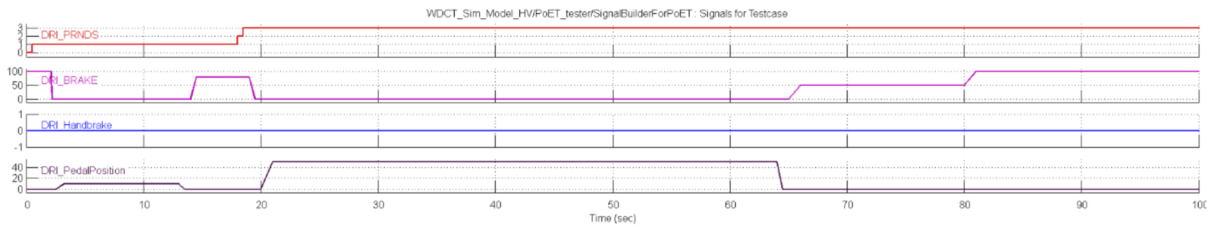


Figure 42: Important signals in the test case for TCU virtual test

Table 1 shows the abbreviation and corresponding meaning and description of those signals in the selected test case.

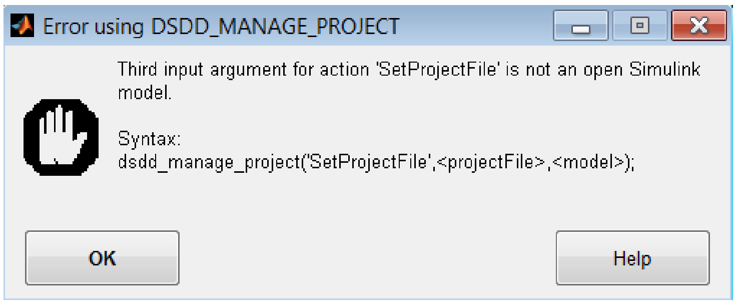
Table 1: Abbreviation and corresponding meaning and description of the signals in the selected test case

Abbreviation	*Meaning*	*Description*
DRI_PRNDS	Driver gear level	0, 1, 2, 3 is “Parking”, “Reverse”, “Neutral”, “Drive” respectively
DRI_BRAKE	Driver brake pedal position	From 0 to 100 [%]
DRI_Handbrake	Driver hand brake	0 means “Off”, 1 means “On”
DRI_PedlPosition	Driver accelerator pedal position	From 0 to 100 [%]

4.6 Test sequence

After both plant model and TCU model can run without errors, the co-simulation is able to be started in MC.

In the test, both the sequential co-simulation with two kinds of sequence and parallel co-simulation are implemented. During the co-simulation, Simulink models bring about a warning (Figure 43). This is because some unnecessary components of TargetLink were deleted in the model but they do not influence the operation of the model.



Error using DSDD_MANAGE_PROJECT

Third input argument for action 'SetProjectFile' is not an open Simulink model.

Syntax:
dsdd_manage_project('SetProjectFile',<projectFile>,<model>);

Figure 43: A warning during co-simulation

4.7 Analysis of test results

4.7.1 Comparison of signals

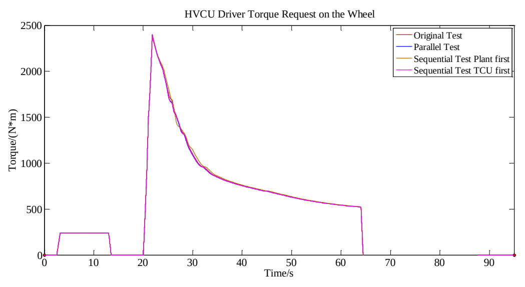
One of the most significant aspects of a co-simulation is that its test results should be close to the classical simulation as much as possible. Thus, the simulation results of parallel as well as sequential tests are verified by comparison with those from the original test. The original test's simulation time step size was flexible and changed during a simulation. In order to get a better and equal comparison, a few rate transition blocks are added in the original test environment and the simulation time step is set to a fixed step size of 1 ms. All comparisons are based on the original test environment with this alteration.

For the comparison, the data are loaded into MATLAB workspace and diagrams are created by a script. On each graph, the same signal of the original test, the parallel test, the sequential test with plant model first or TCU first are displayed.

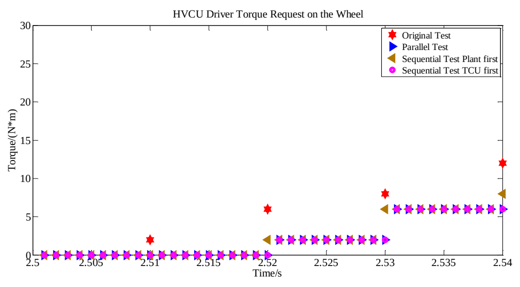
Three typical signals are picked out and discussed because they are able to cover most situations resulting from co-simulation.

The first signal is HVCUDrvrReqTq, which means the estimated HVCU driver torque request on the wheel. It is generated by the simple HCU block in the plant model. HVCUDrvrReqTq's calculation is based on two input variables, acceleration pedal position from the defined test case and the current vehicle speed.

The comparisons between original test and three types of co-simulation via MC can be seen in Figure 44.



(a) Comparison of HVCUDrvrReqTq



(b) A Cut of Comparison of HVCUDrvrReqTq

Figure 44: Comparison of HVCUDrvrReqTq

From Figure 44(a), in most time three types of co-simulation have extremely similar behavior as the original test, but at some time steps there are some small deviations.

In order to investigate details about the time delay problem due to MC, a time cut is shown in Figure 44(b). Before 20 seconds, the vehicle has no speed, so HVCUDrvrReqTq is just based on the accelerator pedal position.

Compared with the original test, Sequential test with Plant first has 10 ms delay. This is because at the initial time step, the signal value is not recorded by the ICOS interface and its sample time in the Simulink model is 10 ms, which means only at every 10 ms this signal value can be updated. Thus, this signal always has 10 ms delay in the sequential test with plant first. Moreover, for the parallel test and the sequential test with TCU first, they need extra extrapolated efforts since the signal is sent from Plant model to TCU. Thus, besides one sample time of the signal delay, one macro time step delay (1 ms) is unavoidable.

To quantify the deviations between the initial test result and co-simulation test results, mean squared error (MSE) is applied. During the entire test duration time, the difference of the signal value between the original test result and the co-simulation test result is compared at each time step. MSE is calculated based on the formula (4.1).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{4.1}$$

where $\hat{Y}$ is a signal value vector of *n* from original test result, Y is the signal value vector of *n* from the co-simulation test result, and *n* is the total number of time steps.

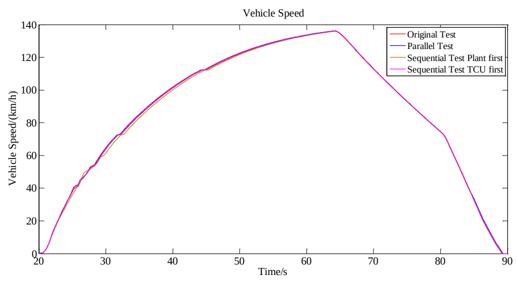
MSE comparison of HVCU Driver Request Torque for three types of co-simulation can be seen in Table 2.

Table 2: MSE comparison of HVCUDrvrReqTq

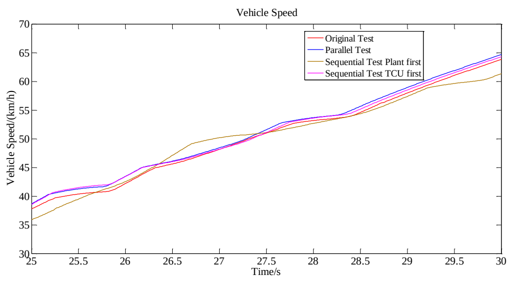
Test type	MSE
Parallel Test	28.24
Sequential Test with Plant first	5.63
Sequential Test with TCU first	28.17

It is obvious that for this signal, the sequential test with plant first has the smallest MSE, due to less extrapolated errors. The input for calculation of this signal is highly reliable on the input accelerator pedal position, which also locates in the plant model, so this signal has comparatively small extrapolated errors from co-simulation in the case of sequential test with plant first. Also, as the plant model calculates first, it has 10 ms delay in comparison with 11 ms delay of other two co-simulations (Figure 44). This is why its MSE is lessened.

The vehicle speed is compared too. In the original test and co-simulation tests, the speed firstly rises to around 135 km/h and then gradually decreases. Some small deviations can be found in Figure 45 and sequential test with TCU first has the biggest deviation.



(a) Vehicle speed comparison



(b) A cut of the vehicle speed comparison

Figure 45: Comparison of vehicle speed

According to the Table 3, by comparing MSE of different tests, it can be seen that the sequential test with TCU first is most close to the original test, the parallel test is a little worse while the sequential test with plant first is the worst. Vehicle speed is affected by multiple parameters and can be a good indicator to show how similar the co-simulation test and the original test are. The sequential test with TCU first would be the best option considering the overall performance.

Table 3: MSE comparison of vehicle speed

Test type	MSE
Parallel Test	0.49
Sequential Test with Plant first	0.73
Sequential Test with TCU first	0.38

The third signal is clutch input speed raw, which is calculated by the CRUISE in the plant model. The comparison of clutch input speed raw is shown in Figure 46. Up to 15 seconds the curves of three types of co-simulation are considerably different from the curve of the original l test.

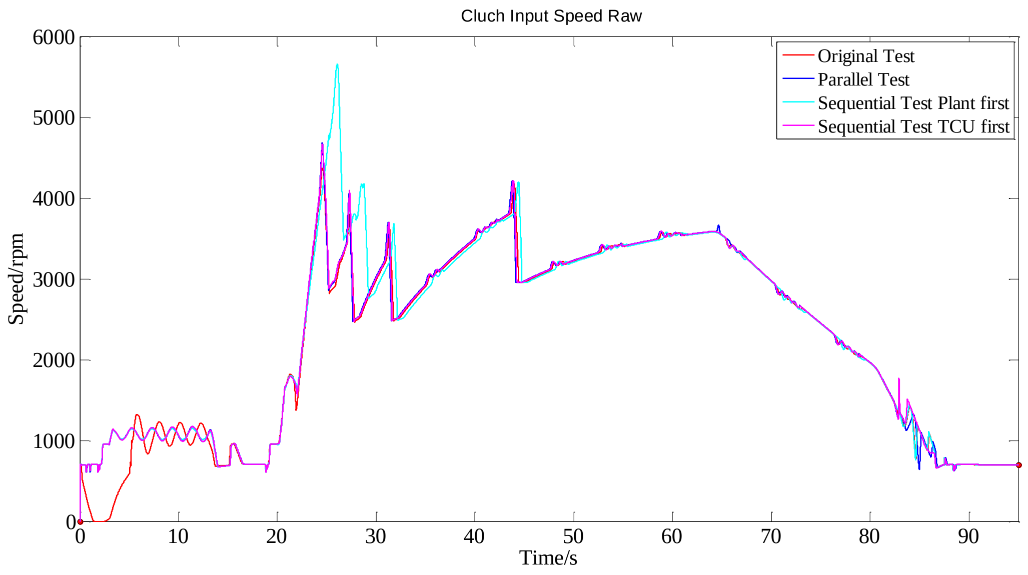


Figure 46: Comparison of clutch input speed raw

After 15 seconds, all curves become overlapped except that the curve of the sequential test with plant first has an apparent larger deviations. The possible reasons why there exists an evident mismatch at the initial stage is to be investigated. One reason might be that at the beginning the vehicle is in the "Reverse" status and some components, which implement the "Reverse", cannot accept any input delay. Another possibility might be that the reversing function of the plant model is damaged after the plant and TCU model are separated.

Table 4: MSE comparison of clutch input speed raw

Test type	MSE
Parallel Test	183.60
Sequential Test with Plant first	369.40
Sequential Test with TCU first	176.78

The MSE comparison also reflects the trend (Table 4). All MSE values are much higher than signals discussed before and MSE of sequential test with plant first has the smallest deviations and only nearly a half of sequential test with plant first. As the value of MSE is dependent on the total variation range as well as the gradient of a signal, it is difficult to find out a universal quantitative criteria in view of MSE for evaluation of the test result quality.

4.7.2 Comparison of computation speed

One attractiveness of the co-simulation via MC is that the calculation speed is probably enhanced. This is because normally a Simulink model only utilizes one core for calculation on a workstation. By contrast, since MC opens several Simulink models in different MATLAB workspaces, utilization of multiple cores calculation on a workstation is achievable. In theory, the computation speed with MC shall be improved.

Table 5 describes initialization time, the calculation time as well as the total time of five different kinds of simulations. The Initialization time of MC denotes the period for co-simulation preparation, including opening a MATLAB workspace, loading some necessary tools and libraries in MATLAB as well as opening a Simulink model. The calculation time is defined as the approximate period from start to stop of a simulation in Simulink (this consists of the implementation of a part of functions in callbacks).

Table 5: Calculation time of 5 kinds of Simulations

Test type	Initialization time/s	Calculation time/s	Total time/s
Original test	Not counted	461	461
Single plant test	Not counted	184	184
Single TCU test	Not counted	406	406
Parallel co-simulation	129	399	528
Sequential co-simulation	136	401	537

To summarize, for a short-time test case (simulation time is 90 seconds in Simulink), until the version of MC used in the thesis, MC does not show any obvious advantage of total simulation time, due to the fact that it needs time to open and initialize MATLAB. However, for a long-time test case, like 20 minutes, the merit of distributed calculation instead of all calculation in one MATLAB might be more apparent, since the actual calculation time of co-simulation (no matter if it is parallel or sequential) with MC is 12% less than it of the original test.

In addition, according to the principle of a parallel co-simulation and a sequential co-simulation, the calculation time of the former should be faster than the latter. In the practical implemented co-simulation, the advantage of parallel co-simulation is not apparent (only 0.5% faster). The most influential reason here is that the calculation time of the single TCU is 2.2 times of the single plant model. (If the efforts of preparation for calculation are eliminated, the multiple is larger). So during the co-simulation, most of the calculation time is contributed by the TCU model so that the time cost of parallel or sequential co-simulation does not show an obvious difference.

4.7.3 Conclusion

To summarize, co-simulation with MC always brings extrapolated errors, no matter if the co-simulation is parallel or sequential. And if a signal's sample time defined in Simulink is larger than the macro time step, then its signal time delay is at least its sample time, as signal has to be updated according to the sample time. This time delay is tolerable in general and has nominal effect on the overall performance.

Considering the subsystem scheduling, there are two possibilities: parallel or sequential. The selection between these two types of co-simulation is a tradeoff. On the one hand, sequential co-simulation has less overall extrapolated errors than parallel type if the triggering sequence of the involved elements is properly defined (otherwise it is even worse). In this thesis, the calculation sequence of the original test is that the TCU calculates first. So in the TCU virtual co-simulation, the sequential test with TCU first， which has the similar simulation sequence as the original test, is the optimal option with the smallest overall extrapolated errors. Therefore, its overall performance like vehicle speed is the closest to the original test.

On the other hand, theoretically the calculation speed of parallel co-simulation is faster than sequential one. However, in this thesis the advantage is not uncovered. This is because if most of the time cost is contributed by one element, while the other elements cost only a small amount of time, then the parallel co-simulation can only take nominal edge on calculation time over the sequential co-simulation.

Considering the total time comparison between original test and TCU virtual test with MC. In this work, there is no benefit of total simulation time due to the initialization time of MC. But its benefit might be more evident in a long-time test case. And in the future version of MC the initialization time might be reduced.

5 Multi-ECU virtual Test

In this thesis, multi-ECU virtual test signifies that more than one ECUs are tested together in a virtual environment. The TCU software, as described in chapter 4, a HCU software and a newly created plant model are connected via ICOS interfaces of Model.CONNECT, and a co-simulation of these three elements is implemented. The significance of this test is that even though the HCU and TCU software is developed on different platforms (MATLAB 2013b and MATLAB 2010a respectively) and developed with different simulation settings, and also developed by different teams, they can be integrated in a co-simulation with Model.CONNECT. So at the early stage of development, the compatibility of two software can be tested instead of finally being tested in a test bed. Thus, front loading can be enhanced. Moreover, if the TCU and HCU is tested in one Simulink model without co-simulation, the calculation speed is comparatively slower since only one core is deployed. Also, each MATLAB workspace has a limited capacity of buffer, so some long-time test cases like NEDC test might not be feasible in a simulation comprising two ECU models as the signals loaded, calculated or saved are almost doubled. Application of co-simulation with distributed calculation is able to solve these problems.

5.1 Original test system introduction

5.1.1 Introduction to the Hybrid Vehicle Control Unit and Hybrid Control Unit

The main task of the Hybrid Vehicle Control Unit is similar to it of a conventional vehicles. It is designed to satisfy the driver demands for different driving situations, under the boundary conditions resulted from the state of vehicle and its components. The HVCU comprises hybrid control software, HV (High Voltage) battery management software and separation C0 clutch actuator control software. These three software modules are developed in parallel and finally generate C code, which is flashed into the real Hybrid Vehicle Control Unit (HVCU) hardware. In this work, the HCU refers to software or Simulink model for the hybrid control and HVCU refers to the real HVCU hardware.

5.1.2 Overview of used programs in original test environments

Various tools are used in original test environments and a brief introduction to them is given.

32-bit MATLAB 2010a is the development and simulation platform of the TCU software. For testing of the TCU model in combination with a plant model, PoET is utilized. PoET, which is based on MATLAB, can generate test cases and preserve test results. It is inserted into the TCU original test environment as a subsystem.

32-bit MATLAB 2013b is the development and simulation platform of the HCU software. In the HCU software development procedure, AVLab is utilized. AVLab is based on MATLAB and enables automatic implementation of MiL as well as SiL testing and generation of test reports. It also offers functions supporting HiL testing.

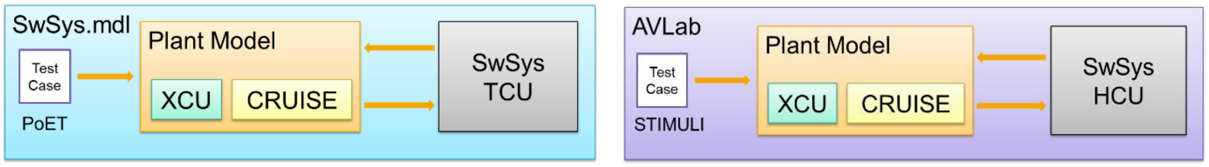
The simulation comprising the HCU model and the plant model is implemented in an AVLab template and scheduled by a GUI of AVLab (AVLab – PoET2). In the template, there is a block for generation of test cases, which is called STIMULI.

The core of the plant model is AVL CRUISE, which simulates the vehicle drive train and dynamics. In both plan models of TCU and HCU, an AVL CRUISE interface to a *.dll file complied from the CRUISE model is inserted.

5.1.3 TCU and HCU original test environment

Figure 47(a) shows the concept of TCU original test environment. The description of it can be found in chapter 4.

The HCU model was tested in a Simulink model (an AVLab template) of MATLAB 2013b, which consists of a structure created by AVLab. In Figure 47(b), the concept of the original test environment of the HCU software is demonstrated.



(a) Original test environment of the TCU (b) Original test environment of the HCU

Figure 47: Original test environment of the HCU and the TCU

AVLab is used to schedule tests, control the process of simulation, save test results and generate test reports. The simulation takes place in an AVLab template. Test cases are provided by STIMULI. The plant model consists of different simplified ECU modules like TCU and an AVL CRUISE interface to the *.dll file from the CRUISE model.

These two software were developed and tested in parallel by different development teams and only when they were flashed into the hardware of a test bed, the performance of their co-operation could be evaluated. But at this stage, it costs more time and money to make changes to the released software and to flash it into the hardware again for an improvement.

Thus, it is significant to create a MiL test environment comprising both control unit models as well as the plant model, in order to achieve early detection of bugs arising from interactions between ECUs.

5.2 Main tasks

The main tasks in this test are:

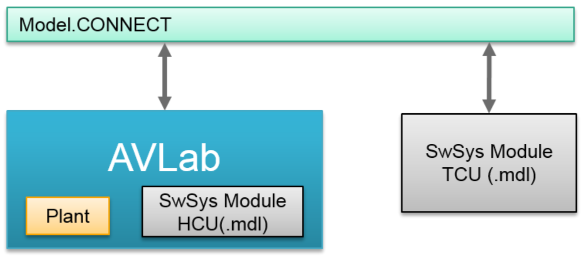
- Creation of new interfaces between the TCU model, HCU model and the new plant model
- Creation of new test cases based on previous test cases
- Modifications and configurations of three models
- MC project setup
- Additional calibration for the hybrid control system
- Validation of co-simulation results

A detailed workflow description can be viewed in the following part.

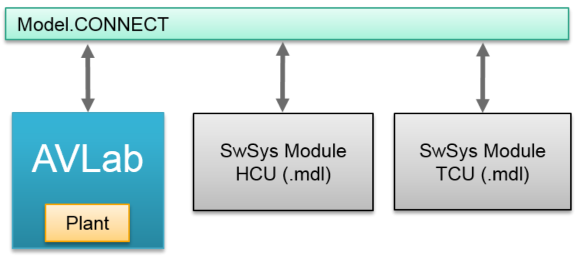
5.3 Test arrangement

5.3.1 Test topologies

There are two possible topologies, which can be seen in Figure 48.



(a) Test topology with two subsystems



(b) Test topology with three subsystems

Figure 48: Test topologies

The first topology is considered on priority, because it has less subsystems so that reduced extrapolated errors are thereby achieved. Meanwhile, it is easier to decide which element calculates firstly in a sequential co-simulation.

However, some blocks of the merged plant model are not compatible with the time step setting of the HCU model, so the first topology is given up and the second three-subsystem topology is utilized in this work.

5.3.2 Workflow

In Figure 49, the workflow of the multi-ECU virtual test is displayed.

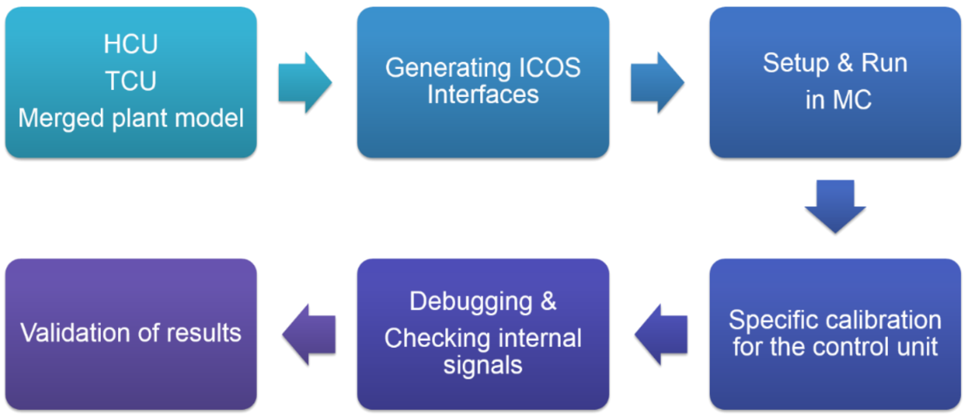

HCU
TCU
Merged plant model
Generating ICOS Interfaces
Setup & Run in MC
Specific calibration for the control unit
Debugging & Checking internal signals
Validation of results

Figure 49: Workflow of the multi-ECU virtual test

Firstly, when the HCU, the TCU and a merged plant model are available, they should be able to run independently without errors. Also, the interfaces (including signal name, physical meaning, data type, sample time etc.) of each model should be defined correctly and correlated with those of others.

The next step is to generate ICOS interfaces for MC in each model, and some modifications are necessary in order to make ICOS interfaces compatible with the original Simulink model, like conversion of datatypes and inserting rate transition blocks.

Once each model of the co-simulation is ready, at the third stage, an MC project should be established, in which all connections between interfaces are created and coupling mechanism as well as the simulation time step are correctly defined. If the MC project can at least run without errors, then it is possible to proceed to the next phase.

However, the running of an MC project does not necessarily mean that the vehicle in the plant model can respond the inputs appropriately. One important reason is that initially all calibratable parameters for the HCU model are default values, so the calibration that is suitable for the new plant model should be implemented before the co-simulation starts. For the TCU model, as the calibration of it is determined by the old plant model, some revisions of the value of calibratable parameters are inevitable.

Subsequently, it is the debugging phase and after each co-simulation the internal signals of HCU and TCU shall be inspected. Therefore, the problem source can be determined (like the plant model, interfaces or calibration) and the improvement to the co-simulation system is able to be achieved.

Finally, if the plant model (simulated vehicle) works as expected, it is essential to compare the co-simulation test results with the original test results for the purpose of validation of the co-simulation test system.

5.4 Modifications of the initial test arrangement

Figure 50 shows what has to be modified in comparison with the original test arrangement. All possible changes utilize the color of red.

An overwhelming challenge is that the original TCU plant model and HCU plant model has a simple HCU and TCU model respectively, therefore the simple HCU and TCU in those plant models should be deleted and afterwards the two old plant models should be merged into one. Moreover, all old interfaces between the control unit and the previous plant model need to be changed, because signals that originally go to the simple TCU or HCU in the old plant model now go to the HCU or TCU model. Meanwhile the new interfaces between TCU and HCU need to be created. Finally, since the TCU and HCU test environment utilizes different tools to generate the test case (PoET and STIMULI respectively), these two kinds of test cases, which provide various inputs, must be combined.

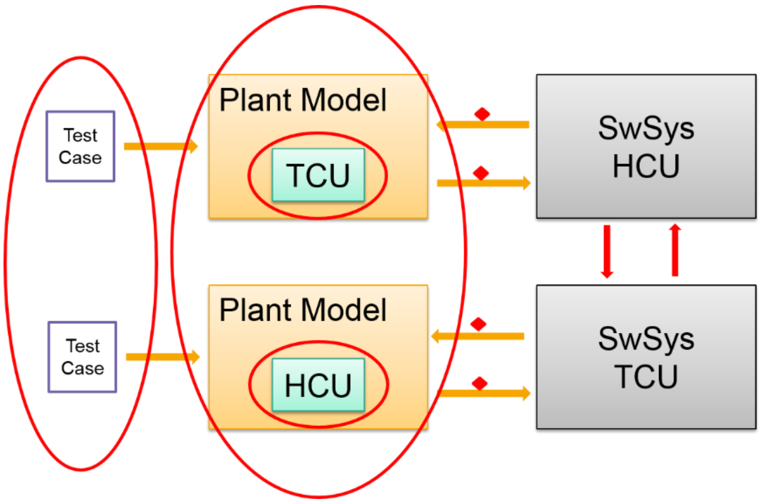


Figure 50: Changes to the original test environment

5.4.1 Creation of new interfaces

The key of creating new interfaces is to find the correlation between different interfaces and reconnect them. Two examples of the correlations can be described in the Figure 51. ASW denotes Application Software.

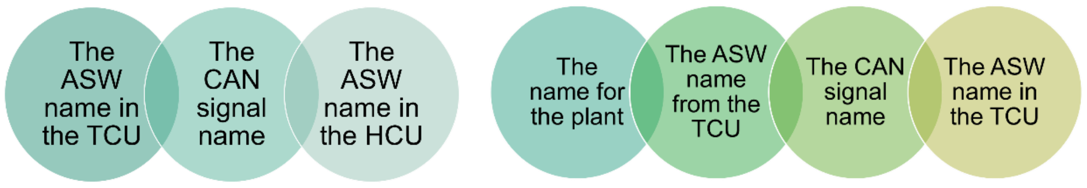


Figure 51: Examples of correlations between interfaces with different names

According to this approach, the new connections between TCU, HCU and Plant model are generated.

5.4.2 Merging two kinds of test cases into one

To merge two kinds of test cases into one, the signals, which have the same physical meaning in both test cases, are kept, like PoET_PosnBrkPedl and DRI_BRAKE, and are named according to the name of the HCU test case. Some uneccessary input signals in the new test are removed. Whether other signals that only appear in the TCU test case or HCU test case shall be kept, is decided by the physical meaning of those signals.

5.4.3 Merging two plant models into one

Merging two plant models into one is accomplished by another colleague, and depends on the confirmed new interfaces. Simply speaking, selected components of those two plant models are copied and put into a new plant model. Meanwhile, the "old" CRUISE model is extended and has more inputs and outputs.

However, this merged plant model was not tested at all after creation and needs to be improved during the further co-simulation.

5.5 Configurations of the Simulink models

The major configurations for HCU, TCU and Plant model are described in this chapter.

5.5.1 HCU model

A frame that comprises HCU model is created to take over the communication with MC. The HCU model lies in SwSys block as a referenced model (Figure 52).

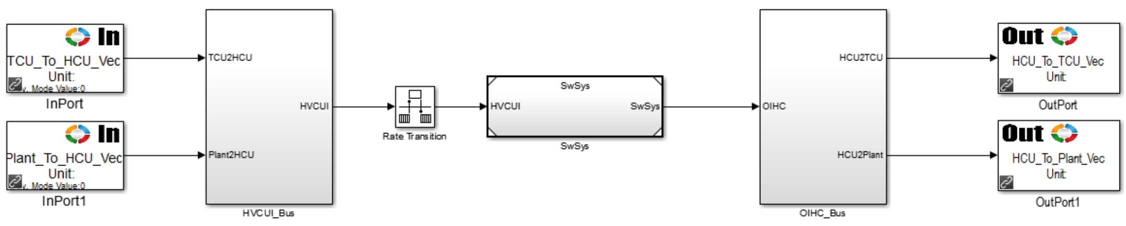


Figure 52: HCU Frame

ICOS OutPort and ICOS InPort blocks are inserted into the Simulink model and the signals transferred by them are processed. Take HVCUI_Bus block as an example (Figure 53), TCU to HCU Vector is separated first and each signal is converted to the data type defined in the bus objective and obtains the same name in the bus. For Plant to HCU Vector, a ver2bus_helper is applied in order to avoid naming manually. After data conversion and naming, all separated signals are combined into one input bus in the defined sequence, which eventually enters the SwSys block.

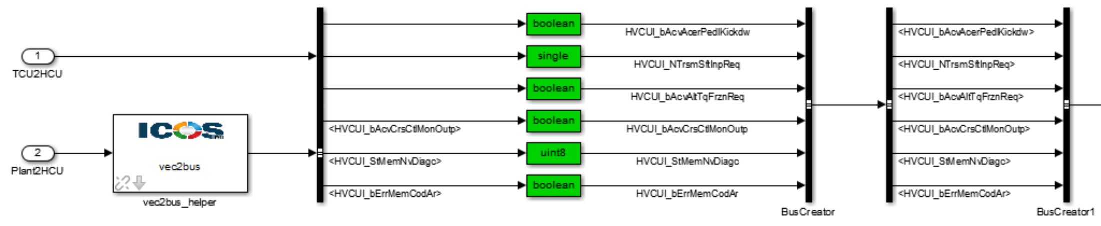


Figure 53: Signal processing of HVCUI_Bus block

Between the HVCUI_Bus block and the model, a rate transition block is still necessary to ensure a consistent data transfer to systems with different cycle times. The output signals in the OIHC_Bus block are processed similar to the input signals.

5.5.2 TCU model

Similarly, the TCU model from TCU virtual co-simulation is modified in line with the new interfaces. In comparison to the model of TCU virtual co-simulation in chapter 4, two new interfaces, HCU_To_TCU_Vec ICOS InPort and a TCU_To_HCU_Vec ICOS OutPort, are inserted (Figure 54).

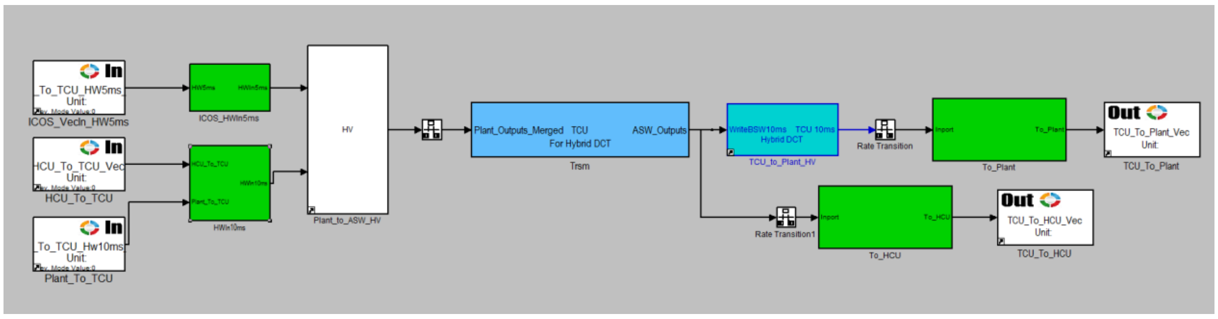


Figure 54: Changes to the TCU model

5.5.3 Plant model

The Plant model lies in the SwSys_Test model, which includes a framework created by AVLab. Figure 55 shows the full structure of SwSys_Test model. AVLab environment is utilized because the co-simulation is expected to follow the previous workflow and tools without additional efforts. Furthermore, with AVLab, automatic generation of test reports is attainable.

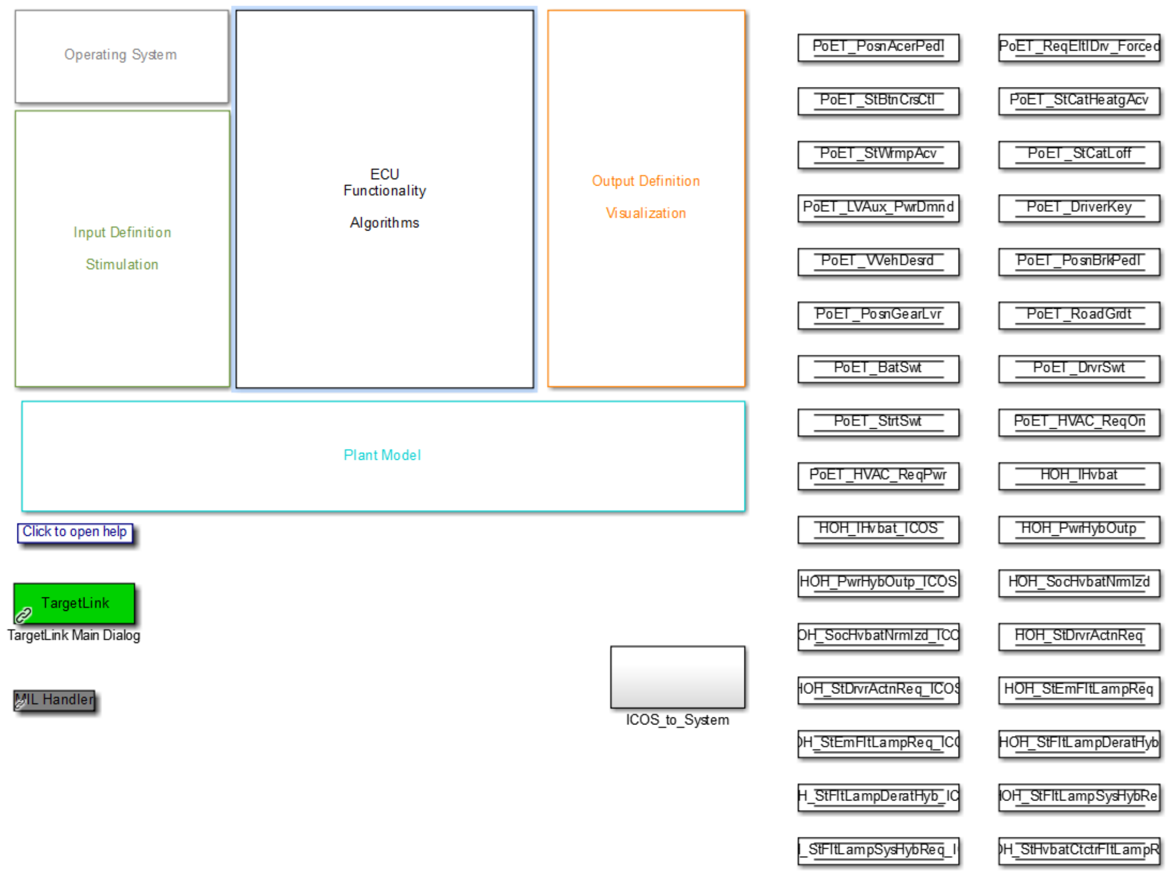


Figure 55: Plant model test environment

Many changes are made in the SwSys_Test model. Instead of integrating the HCU as a referenced model into the lowest level of the ECU functionality block, the interfaces to MC are required here.

In Markus Kogler's first attempt [34], instead of referencing the HCU model, ICOS OutPorts and ICOS InPorts, which were intended to send the signals to the corresponding counterparts within the frame, were inserted into ECU Functionality block. However, this try could not work in MC co-simulation due to its conflicts with the function of call subsystem in the SwSys_Test model. Thus another solution is utilized. The data exchange between MC and Simulink model is shifted to the highest level. At the lowest level, the signals are stored in further DataStore blocks (Figure 56) and transmitted to the corresponding DataStore blocks in the newly created subsystem ICOS_to_System at the top level, in which data exchange using ICOS InPorts and OutPorts takes place (Figure 57).

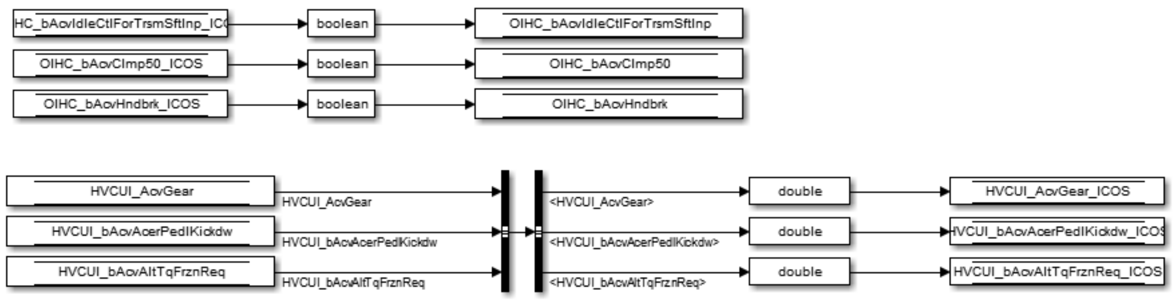


Figure 56: DataStore blocks at the lowest level

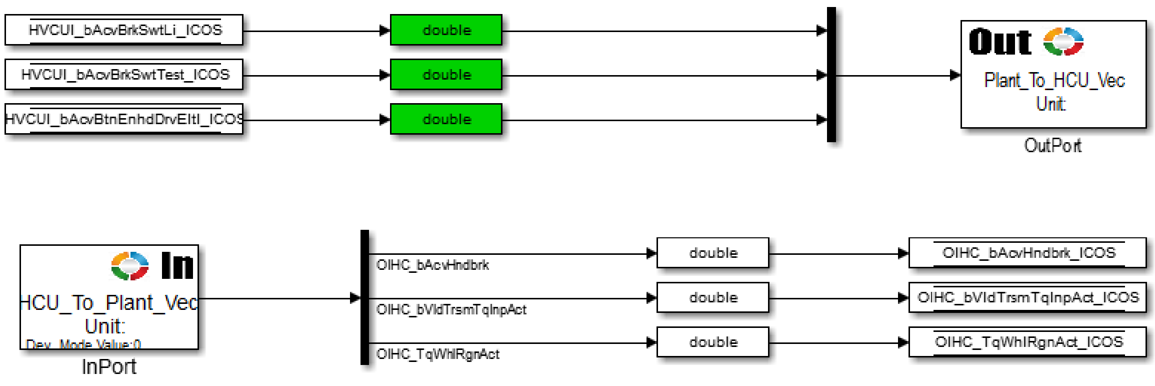


Figure 57: DataStore blocks and ICOS interfaces at the top level

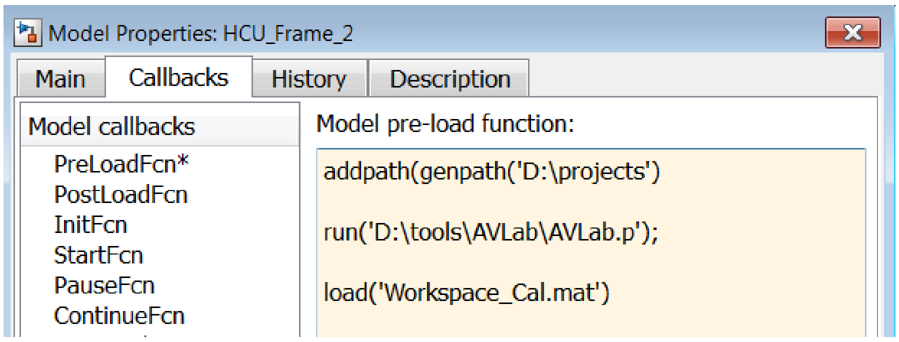
The disadvantage of this configuration is that it doubles the quantity of the memory blocks since there is now another DataStore block for each signal in the call subsystem. The original memory blocks are copied and a suffix "_ICOS" is supplemented to the new signal names, so that it is easier to find out the data flow path of each signal.

All interfaces, data-transmission blocks and renaming are implemented automatically with MATLAB scripts to avoid manual errors.

5.5.4 Additional functions in callbacks of Simulink models

In TCU Simulink model, the callback setting is as same as the setting in TCU virtual test.

In HCU Simulink model, the callback setting can be found in Figure 58.



```
addpath(genpath('D:\projects')

run('D:\tools\AVLab\AVLab.p');

load('Workspace_Cal.mat')
```

Figure 58: Callbacks of HCU Frame model

In the callback, it is necessary to add the project path firstly. Then the AVLab is activated because only with AVLab all parameters in the workspace are accessible. Finally, all workspace variables, the calibration from the vehicle as well as some manual modifications of calibratable values are loaded from the Workspcae_Cal.mat.

In the SwSys_Test model (plant model), the setting is similar to the HCU model. In addition, if ICOS embedded is decided to be used, in the initial function of the callback the command that activates MC shall be supplemented.

5.5.5 Specific calibration of the control unit

In the HCU and TCU model, there are a large number of calibratable parameters. For example, NOSC_UDcdcHvThdForPowup_P (high side voltage threshold for transition from wake-up to power-up of the HVCU). This parameter has to be calibrated appropriately. If the threshold is too large, then in certain conditions the driver is not able to start the vehicle. On contrary, if the threshold is too small, the vehicle might work at risk.

With continuous update of the HCU software, the original calibration of the HCU MiL test becomes invalid. Therefore, the calibration used in the real vehicle is adopted here, but it needs to be adapted for the new plant model. The "old" calibration of the TCU software is adopted but the problem is also critical, since it has not been updated with the higher version of HCU software, either.

The calibration work in this thesis is mainly focused on the HCU. Through testing and inspecting the internal states and signals of the HCU model, additional calibration is required to ensure that the HCU model can work smoothly.

5.6 Model.CONNECT Setup

The time step of each MATLAB Wrapper is set to 10 ms. The TCU model is executed by MATLAB 2010a while the HCU and the Plant model are executed by MATLAB 2013b. Different MATLAB Wrappers are connected via vectors (Figure 59). To avoid the trouble of deciding the triggering sequence of three subsystems, the parallel co-simulation is adopted.

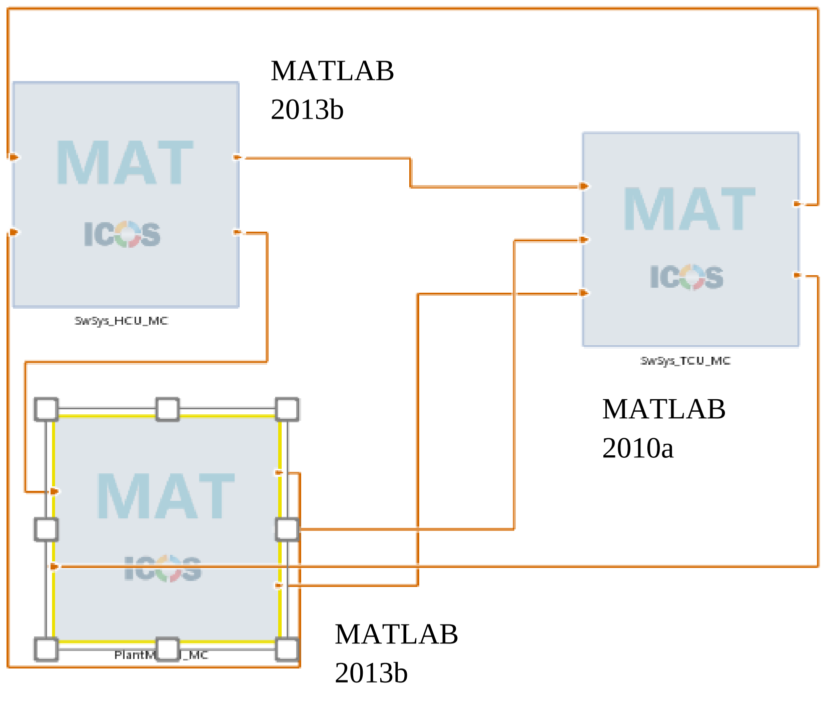


Figure 59: Model.CONNECT setup of multi-ECU virtual test

5.7 Test case

For the further validation of co-simulation results, a special test procedure is used. The applied test case 12 (Traction Torque Demand Determination) includes two acceleration phases, two deceleration phases and ends with a stop of the vehicle. It is created by merging the previous test cases from two simulation environments.

5.8 Test sequence

There are two types of test sequences. One is the standard co-simulation started by MC. Another is ICOS embedded mode started by AVLab. The second one is more interesting because it is possible to integrate MC into the existing tool chains and development procedure. The workflow of the second test sequence is shown in Figure 60.

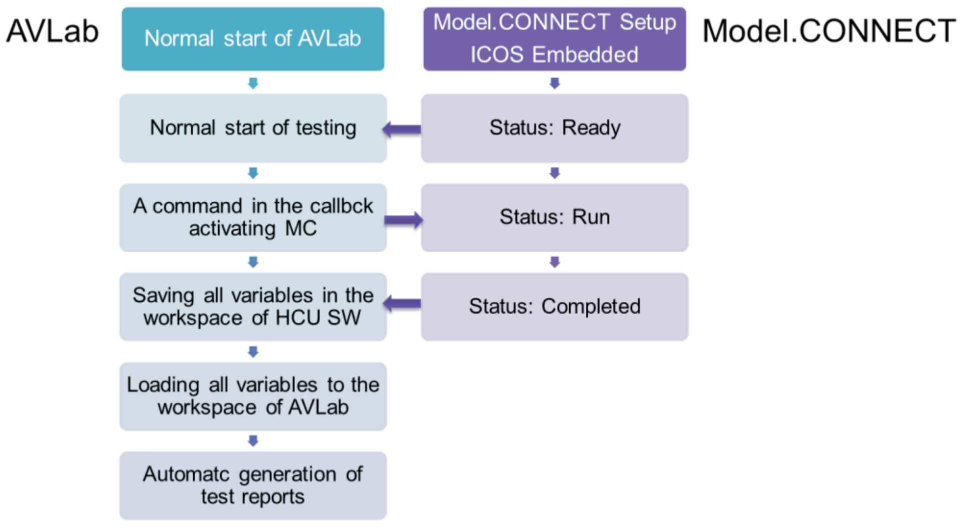

AVLab
Normal start of AVLab
Model.CONNECT Setup ICOS Embedded
Model.CONNECT
Normal start of testing
Status: Ready
A command in the callbck activating MC
Status: Run
Saving all variables in the workspace of HCU SW
Status: Completed
Loading all variables to the workspace of AVLab
Automatc generation of test reports

Figure 60: Workflow of ICOS embedded mode with AVLab

For the second solution, appropriate operation of ICOS embedded mode and callback in the Simulink model is necessary. In the version of MC R2017f Build 091, these two requirements can be fulfilled.

5.9 Test results

MC can run the project without errors. After some efforts on change of the initial state of the involved systems and updating the calibration for HCU, the simulated vehicle in the plant model still cannot move. An apparent problem of engine speed is detected at the beginning of the operation (Figure 61).

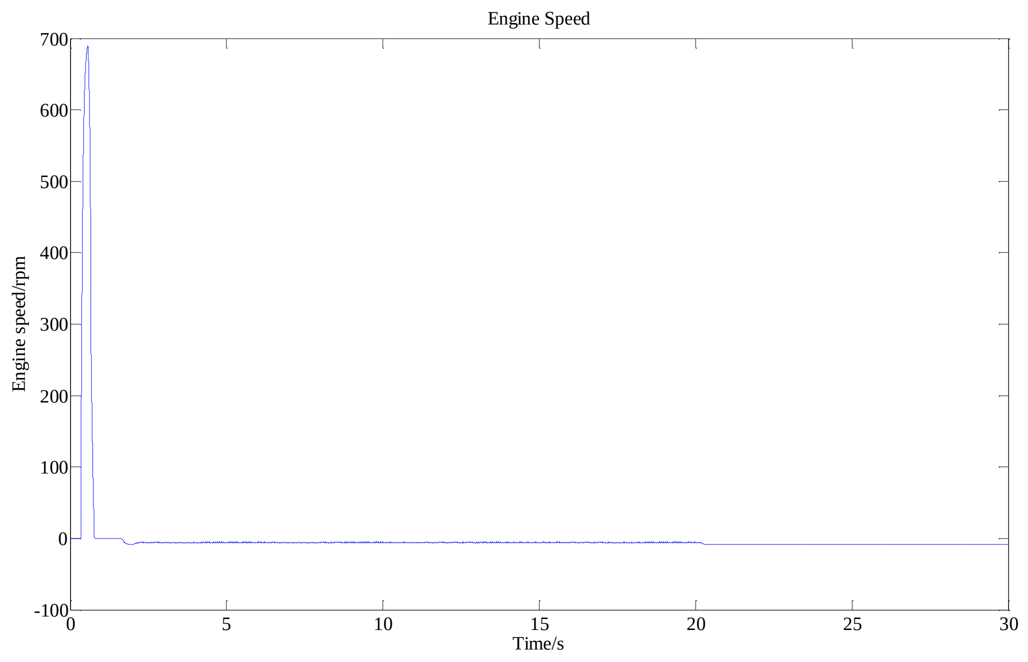


Figure 61: A problem of the engine speed

It is clear that during the engine start-up, the engine speed suddenly drops in a few time steps and then the engine is stalled.

The most possible reason for the engine speed stalling behavior is caused by wrong calibration of control units. However, the possible problem could also derive from the definition of interfaces, inappropriate coupling between the plant model and control units, and incorrectness of the plant model. Another trouble is that the definition of certain signals are different in the HCU model and the TCU model. For example, in the TCU model the physical value of the neutral gear is 0, while in HCU model it is 12. Some of them might not be synchronized.

To solve this problem, the control path of the start-up function needs to be further investigated.

5.10 Conclusion

In this chapter, the new interfaces between the TCU model, HCU model and a new plant model are created. After modifications to the original models, co-simulation can run under the framework of MC. Based on the co-simulation results, analysis of the hybrid control system can be achieved. Thus, the concept of multi-ECU virtual test proves to be feasible. Also, the co-simulation can integrate AVLab into the simulation procedure, which enables the utilization of existing reporting functions and tool chains.

However, due to various difficulties including the calibration of control units, coupling of models or feasibility of the new plant model, the simulated vehicle has a problem of engine stalling. In the future work, this problem can be solved by inspecting internal signals of start-up control blocks.

6 HVCU real test

6.1 Description of the basic platform

6.1.1 Introduction to the HVCU

The real control unit HVCU comprises not only hybrid control software, but also HV (High Voltage) battery management software and separation C0 clutch actuator control software. The hybrid control software, is the core function for a hybrid vehicle. Based on driver inputs, HCU can implement various control strategies and tasks, including torque-split, decision-making of engine start/stop and performance optimization. The HV battery management software is mainly used for calculation of SOC (state of charge) based on voltage and current of each cell. The separation C0 clutch actuator control software is used to separate or close the clutch, or decouple or couple the engine. It has a feature of torque tracking, which means that the hydraulic pressure controlled by C0 clutch control software is dependent on the actual engine torque output. This feature can save energy.

The architecture consisting of three software was chosen since the C0 control software requires only a small amount of computational efforts, but many hardware interfaces, in contrast, BMS and HCU software requires a high computing resources.

There are three layers in the software in HVCU: BSW (Basic Software) for signal processing, which provides drivers for the hardware, CIL (Customer Interface Layer), which can be viewed as a middleware, and ASW (Application Software), which contains application software as well as level 2 monitoring (Figure 62).

The data exchange between C0 and HCU, like the status of the clutch or a torque request, takes place via the interface. Also, between BMS and HCU there is a data exchange about the SOC (state of charge) of the HV battery and other parameters.

Furthermore, HCU, C0 as well as BMS communicates with other control unites via CAN messages. For example, the BMS requires the individual cell voltages of the HV battery and calculates the SOC for the individual cells, which is sent via CAN to other control units. These interactions must be considered here, since they internally take place in a real control unit in a vehicle and the fulfillment of the functions of a vehicle cannot live without them [34].

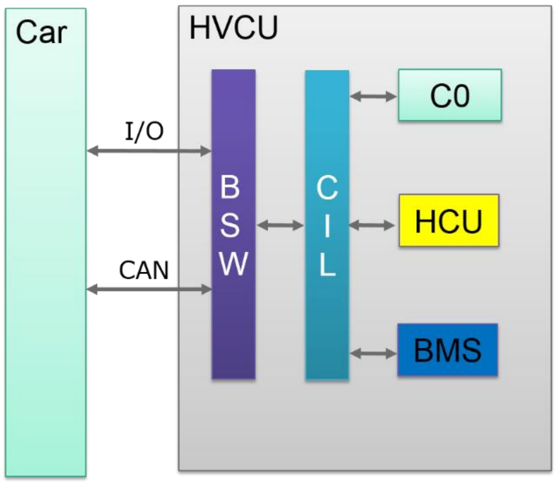


Figure 62: General HVCU Software Architecture [34]

Furthermore, the HVCU communicates with other control units in a vehicle via three CAN buses consisting of HYB CAN (Hybrid CAN), PT CAN (Powertrain CAN) and PVT CAN (Private CAN). In addition, a LIN bus is connected with an intelligent battery sensor (IBS), which monitors the voltage and current of the conventional vehicle battery (Figure 63).

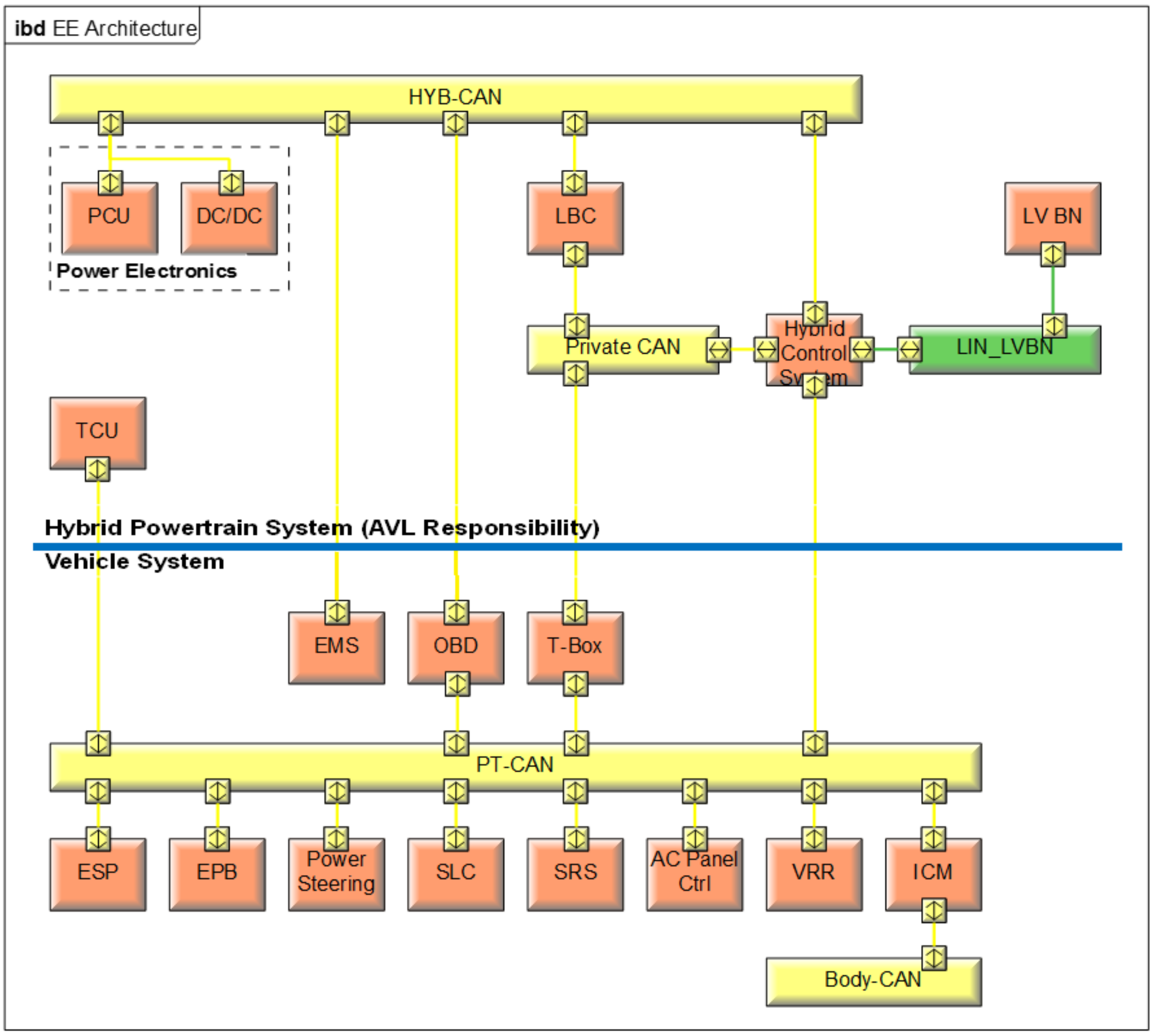


Figure 63: Connections between components and CAN buses [42]

Moreover, some components are connected with each other via a line in a signal loop HVIL (High Voltage Interlock Loop) (Figure 64). The HVCU is the starting point as well as the end point. It applies a voltage of 5 volts, and measures the voltage level of the incoming signal. The measured voltage drop can only be in a certain tolerance band, otherwise the diagnosis will be activated.

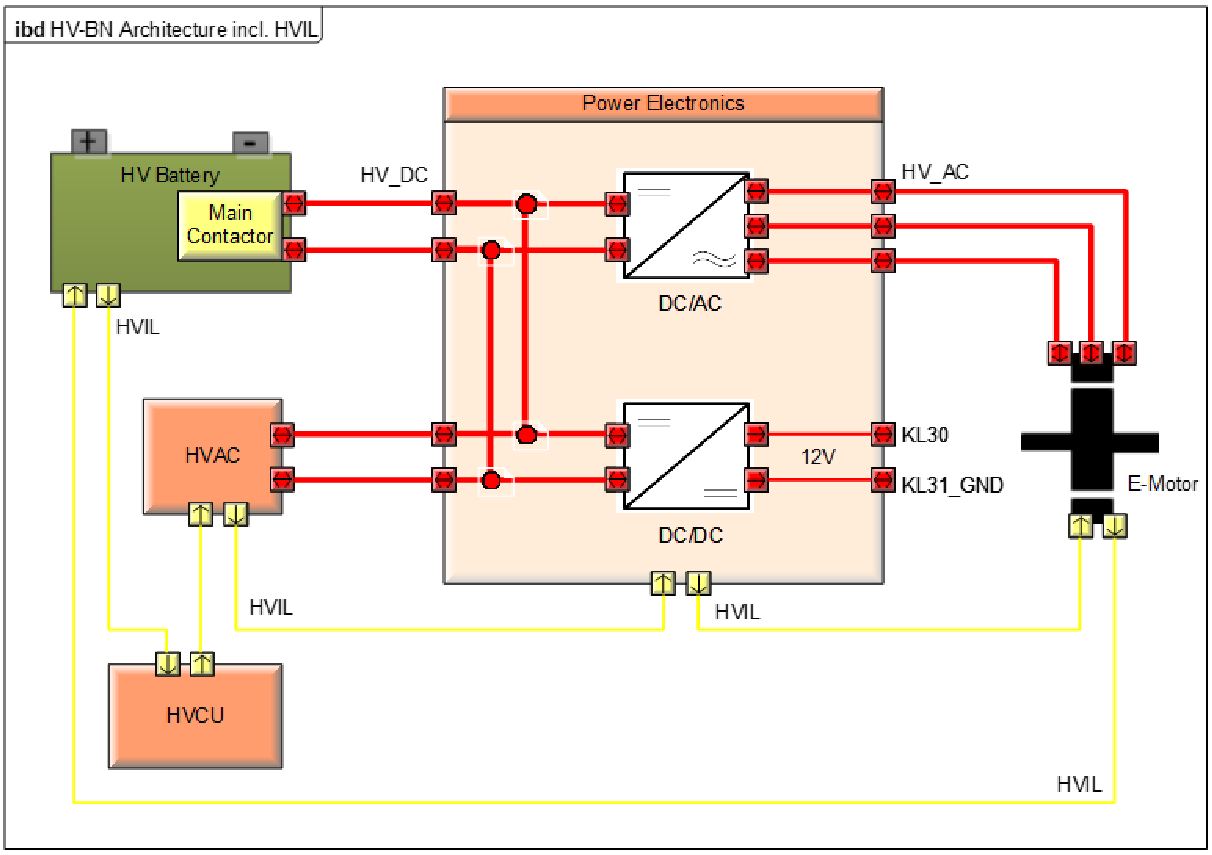


Figure 64: Components monitored by HVIL signal [42]

This signal checks whether the wiring of the power electronics, the electric motor, the HVAC (Heating, Ventilation and Air Conditioning) and the HV battery is connected.

The control unit also has interfaces with sensors and actuators. The HCU software needs two digital outputs, HybMainRly_Ctrl (Hybrid Main Relay Control) for the main relay and EngCrkRq_Ctrl (Engine Crank Request Control) for the transfer of the signal from clamp 50 (crank signal), and a few inputs (Figure 65). With clamp 50 signal, the starter information is passed on to the starter. If the driver wants to start, this information is first transmitted directly to the HVCU, which then decides whether to drive electrically or relay the signal to the internal combustion engine control unit.

Most other pins, about 20 in number, are required by the BMS or C0, which results in increasing complexity of the test setup. Additional analog signals for temperature and pressure sensors as well as for switches with some resisters are required in series. Besides, several digital inputs and outputs for relays of the HV battery, switches and controls of solenoid valves are necessary in a real HVCU test environment [34]. The connected components of HVCU are shown in Figure 65.

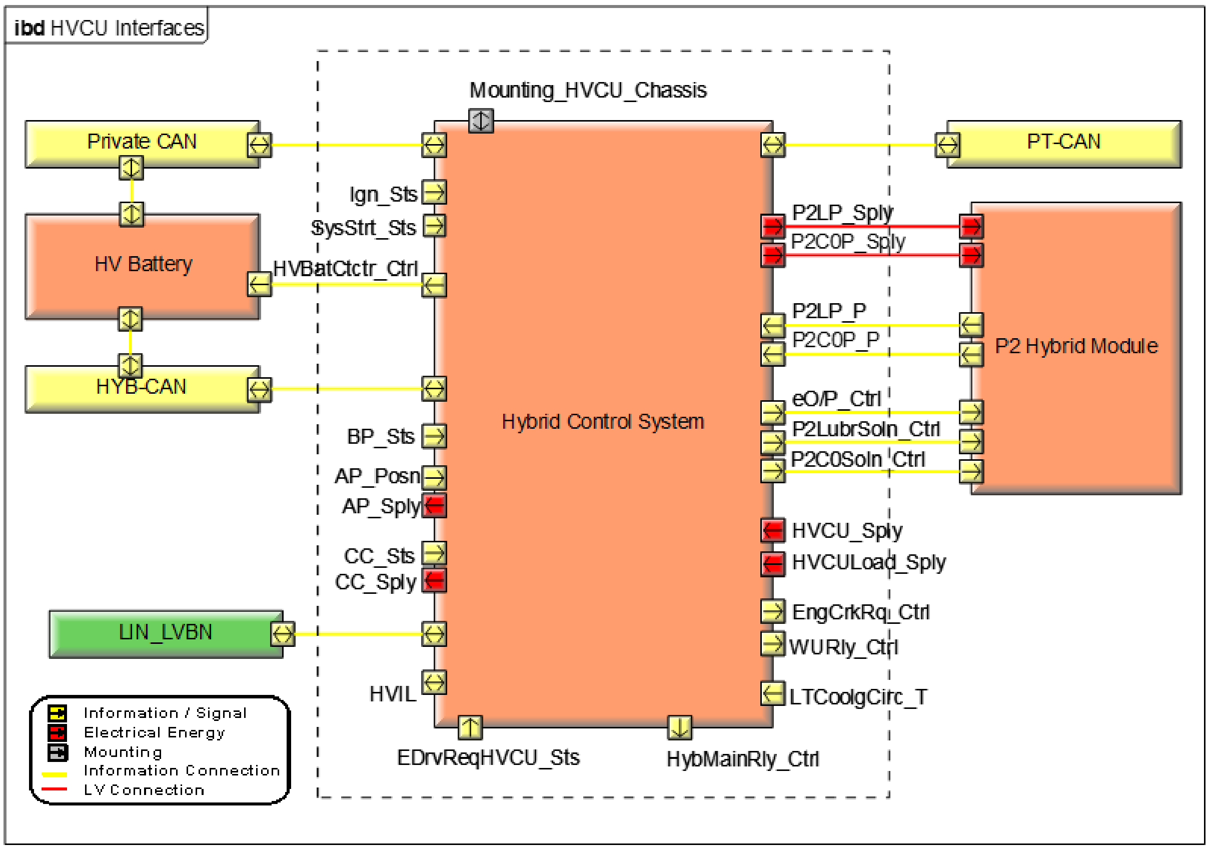


Figure 65: HVCU and connected components [42]

6.1.2 Test topology

MC serves as the interface between the plant model and the HVCU hardware, as shown in Figure 66. Via ICOS interfaces, the data can be transferred between Simulink model and MC, while CAN messages can be transferred between MC and the HVCU as well as I/O hardware using ICOS RealTime blocks.

In comparison with a real car, now in the HVCU real-time co-simulation, some additional components (like pressure sensors, brake pedal sensors, batteries, etc.), which feed signals back to the control unit, also need to be simulated. Therefore, some extensions for the plant model of HCU are necessary.

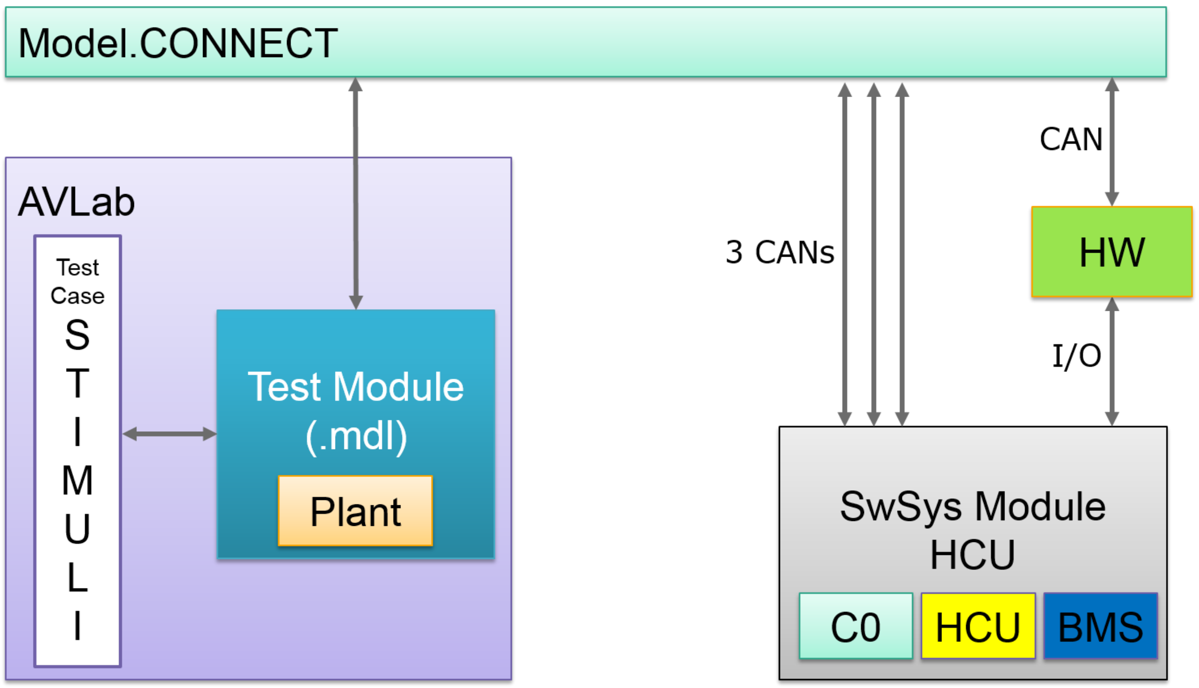


Figure 66: MC as the interface between the plant model and the control unit [34]

6.2 Configurations of the basic platform

The setup and configuration of the basic platform was accomplished by Markus Kogler [34]. In the following parts a summary of the previous work is presented.

6.2.1 Simulation of the interfaces of the HVCU

In order to correctly implement a test, all hardware interfaces of the control unit have to be supplied with the corresponding signals. Since most real components in a vehicle are not available in the office, like a real accelerator pedal or pressure sensors, the inputs and outputs of the HVCU have to be supplied or received by modules in the plant model. Table 6 shows a part of the important inputs and outputs of the HVCU.

Table 6: Part of important pins of the HVCU [43]

Connector Pin	*Name*	*Description*	*Application max current*
A2	C0 solenoid LS drive and Reserved solenoid LS drive	ON/OFF Low side outputs for current control return	3A
A4	C0 solenoid HS drive	PWM high side outputs with current control	1.5A
A11	Crank signal(KL50)	Digital signal input	
A17	P2 lubrication solenoid LS Drive	ON/OFF Low side outputs for current control return	1.5A
A19	P2 lubrication solenoid HS drive	PWM high side outputs	1.5A
A20	Precharging relay HS Drive	ON/OFF High side outputs	700mA
A23	Wake up control relay	ON/OFF Low side switch output	100mA
A25	Brake pedal switch	Digital signal input	
A40	Brake pedal lamp	Digital signal input	
A42	Focus_E_Drive	Digital signal input	
A43	Ignition(KL15)	Digital signal input	
A50	HV Battery negative relay HS drive	ON/OFF High side outputs	700mA
A53	Hybrid main relay	ON/OFF Low side switch output	100mA
K11	Acceleration pedal sensor 1 signal	Analog signal input	
K15	P2 LP_P1 Sensor signal	Analog signal input	
K37	P2 LP_P2 Sensor signal	Analog signal input	
K55	Acceleration pedal sensor 2 signal	Analog signal input	
K57	Cruise control button status signal	Analog signal input	
K59	P2 LP_P3 Sensor signal	Analog signal input	
K80	Low temperature sensor Signal	Analog signal input	

A few signals are only simulated by external circuits. For example, cruise control switch and the NTC (Negative Temperature Coefficient) temperature sensor of the cooling circuit are analog inputs to HVCU. They are generated by an external voltage divider, which is powered by the HVCU power. Most input and output signals are simulated or processed by MicroMods by the means of AD/DA conversion or CAN-digital conversion.

6.2.2 PEAK modules (MicroMods)

Since I/O signals also need to be sent and received by MC, it is necessary to take advantage of modules, which can communicate via a CAN bus.

After comparisons with other companies' products like National Instruments modules or MRS Electronic, I/O modules manufactured by the company PEAK-System were selected. By combining three modules, one PCAN-MicroMod Mix for digital I/Os and two PCAN-MicroMod Analog for analog signals, the entire interfaces of HVCU could operate [34].

6.2.3 Simulink setup

Similar to the setup of SwSys_Test of HCU in Multi-ECU virtual co-simulation, the ICOS interfaces were created and the data types were converted.

One big difference is that the signals from the plant, which were connected to analog inputs of the control unit, had to be converted into a 12 bit signal in Simulink. One example is the two voltage signals for the accelerator pedal. Also for other I/O signals, like the temperature sensor of the cooling liquid (coolant temperature) as well as the cruise control switches, this kind of conversion existed [34].

6.2.4 Model.CONNECT setup

The first attempt of the architecture of MC can be viewed in Figure 67. Since ICOS interfaces transferring vectors were not utilized and in the plant model there were more than 400 ICOS InPort and OutPort blocks. Each ICOS interface block is an S-function. Thus, in the plant model not only the vehicle reactions had to be calculated, but also an S-function was executed for each ICOS interface. Due to the aforementioned reason, the calculation time of the Simulink model was apparently higher.

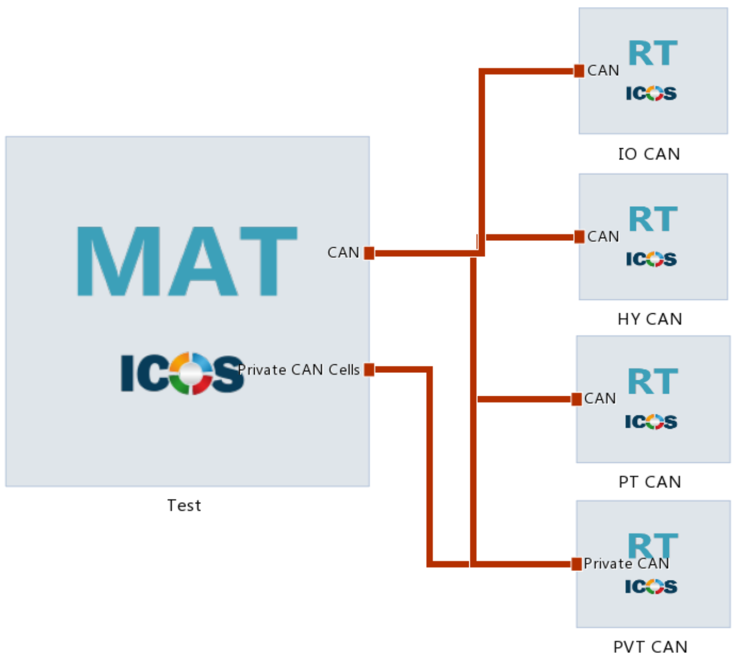


Figure 67: Architecture of the real test in MC [34]

A solution to this problem was found. Computation time can be distributed to several cores by supplementation of another Simulink model, which worked for the Mux and Demux function (Figure 68). The signals from or to the SwSys_Test were combined into two vectors and written or read over two ICOS blocks. In another Simulink model one vector was read and split into single signals or single signals were combined into one vector. This model (Mux_Demux) was installed after the plant model in MC. This setup had the advantage that one core calculated the plant model and another served many ICOS interfaces (Figure 68) [34].

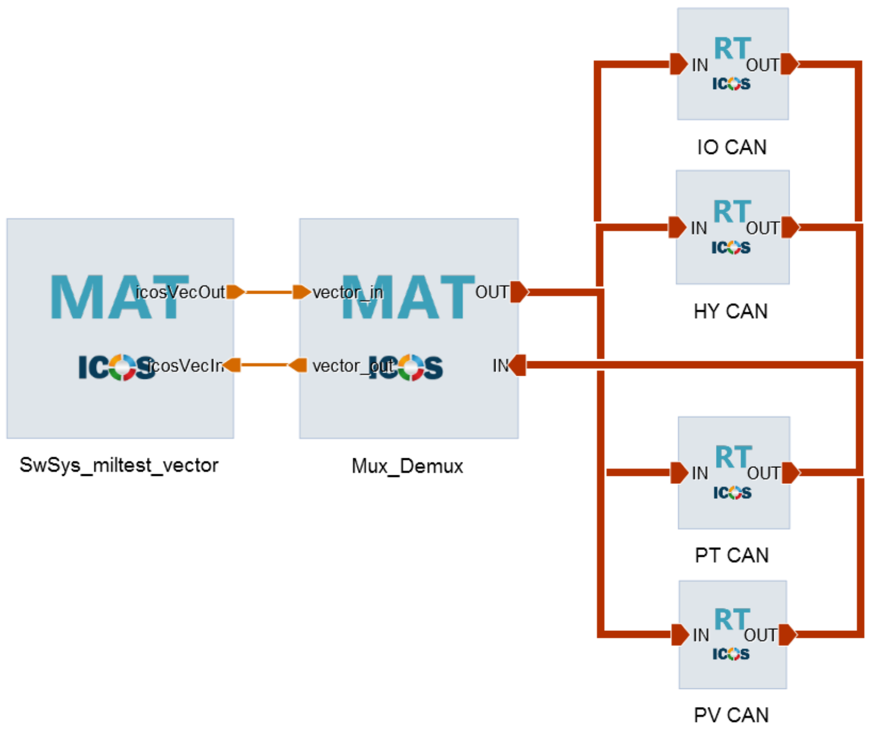


Figure 68: Architecture in MC with an additional model for vector extraction to individual signals [34]

6.3 Initial state of the test and challenges

One test case was implemented and the internal signals as well as CAN signals of the HVCU were measured by a CANape. One problem was that with a manual calibration in CANape the vehicle in the plant model could react, however, the HVCU was unable to close or separate the clutch properly. And the function of the control unit was not tested on the basic test platform, thereby it is unknown whether all interfaces are correctly set up.

Another challenge was that the offline system ran slower than real-time, which could not exactly cooperate with the real-time hardware HVCU. Also, as HVCU's complexity has considerably grown with continuous development, the management of a large number of ICOS interfaces and correct links between interfaces has to be concerned.

The difficulties of a general HiL test system is also faced with in this thesis. The main challenges for this work can be summarized as MC-related issues and general HiL-related issues:

- ❖ MC-related issues: ICOS interfaces, connections and real-time condition.
- ❖ General HiL-related issues:
  - ECU: update and calibration.
  - I/O simulation: AD/DA conversion, external load design, sensor simulation.
  - Plant model: update, parameterization, interfaces and calculation speed.

6.4 Main improvements to I/O simulation

6.4.1 Clutch separation and lubrication control loop

To fulfill the control of clutch separation and lubrication function of the HVCU, supplementary hardware and plant model configurations are needed.

Figure 65 shows the signals between the HVCU (hybrid control system) and the clutch module C0 (P2 Hybrid Module) on the right. On the left side are the signals for the contactors of the HV battery, which were already appropriately configured.

The functional principle for the hydraulic clutch control is shown in Figure 69. An electric oil pump builds up the line pressure and lubrication pressure. In view of C0 clutch control, the current through the C0 solenoid in the 3/2-way valve CLUTCH VFS (right in Figure 69) indicates the demand of pressure for the clutch. In light of lubrication control, part of pressure from the oil pump is used for LUBE (lubrication of C0). The LUBE SPOOL VALVE 2/2-way valve releases lubrication flow depending on the control pressure, which is based on the line pressure and the current through the 3/2-way valve LUBE MDA NH (in the middle of Figure 69) [34].

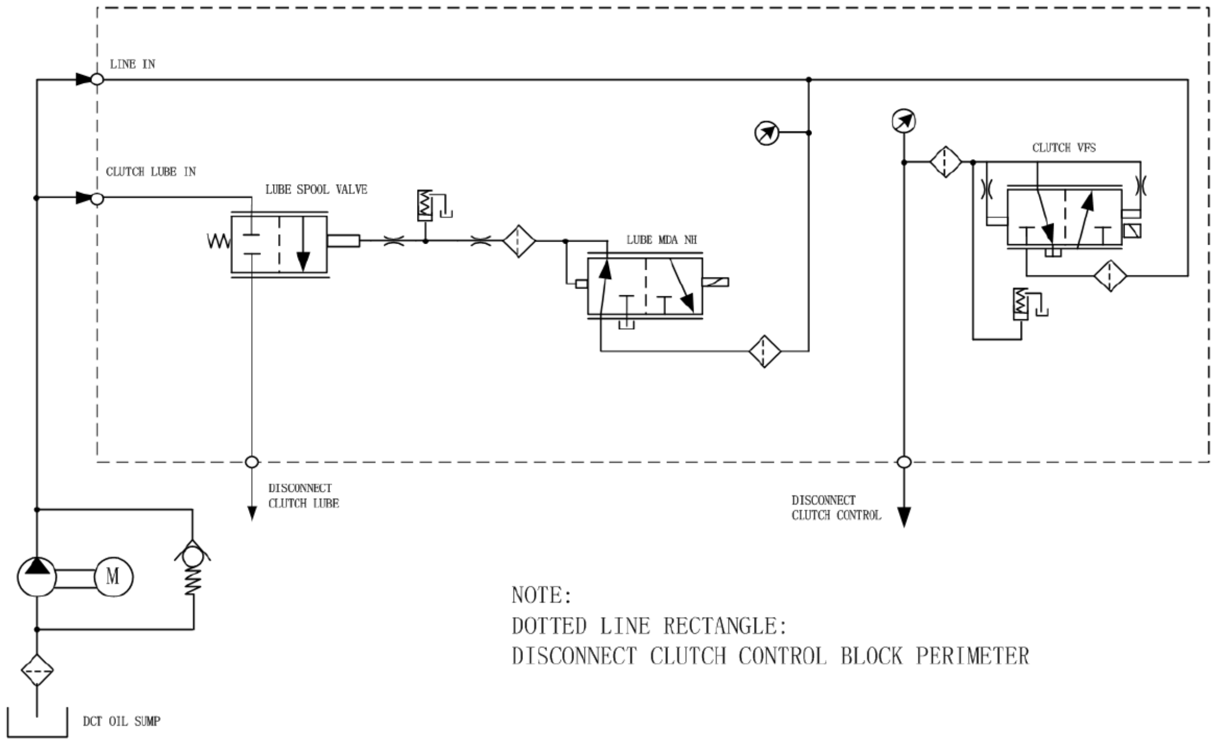


Figure 69: Hydraulic circuit diagram of the clutch control [42]

The two valves are controlled by two pulse-width-modulated (PWM) signals respectively and connected as real actuators to the control unit. In a real vehicle the liquid pressure is measured by a pressure sensor, which feeds back the signal to the HVCU as an analog input. Although two real solenoid valves are connected with the HVCU in the real test, it is not possible to install a real oil pump and therefore measure the pressure by sensors in office. Thus, the built-up pressure has to be simulated. The current through the solenoid is measured and converted into a corresponding pressure using a characteristic curve, after which the simulated pressure sensor signal as an analog input (0 – 5 volts) is given back to the HVCU.

The full control loop of the clutch separation control is shown in Figure 70. The principle of the lubrication control is almost same only with different characteristic curves.

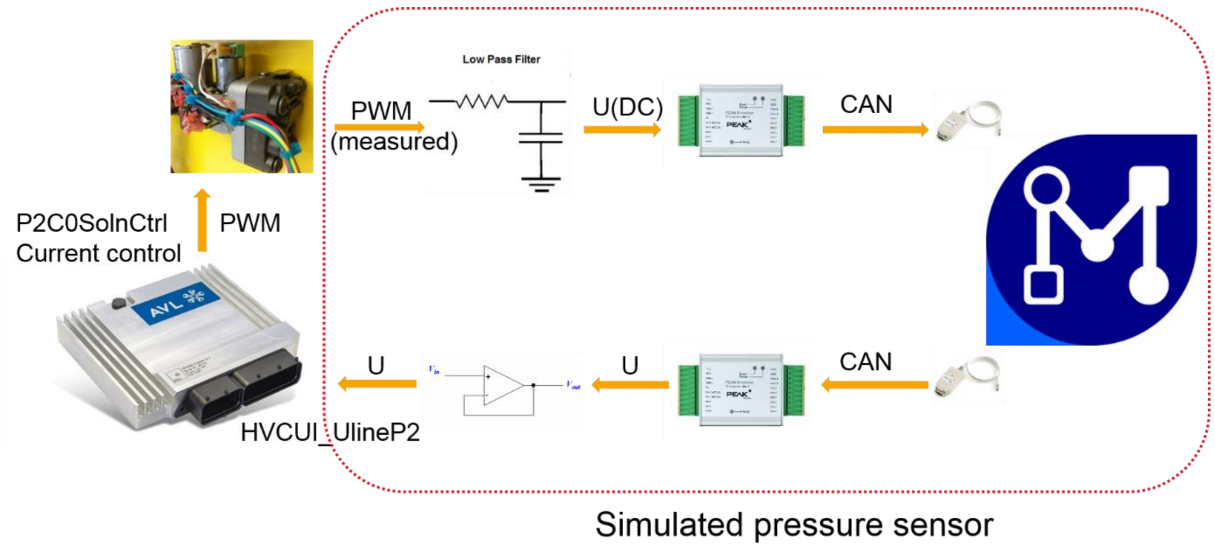


Figure 70: Control loop of the clutch separation

A PWM (pulse-width-modulated) signal (P2C0SolnCtrl) is outputted from the HVCU for the control of the solenoid valve in the hydraulic module of the C0 clutch. The PWM's low level and high level is 0 volts and 12 volts respectively and its frequency is 3000Hz. By changing the duty cycle of the PWM, the current through the solenoid can vary from 0 to 1.5 amperes.

Firstly, the PWM signal through the solenoid is measured and then rectified by a second-order low-pass filter (Figure 71), after which it is converted into an approximate direct current.

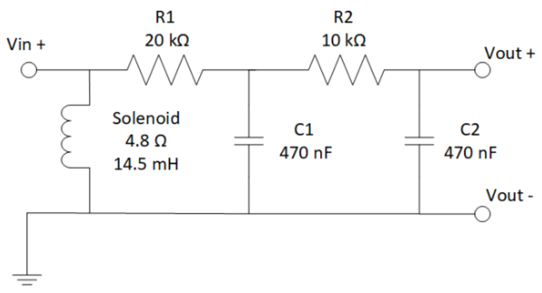


Figure 71: A second-order low-pass filter

The Low Pass filter circuit only enables the low-frequency signals from 0 Hz to its cut-off frequency, f_c , to pass, while blocking other signals with higher frequency. f_c is calculated according to the formula (6.1).

$$f_c = \frac{1}{2\pi RC} \tag{6.1}$$

$f_c\dots$	The cut-off frequency
$R\dots$	Resistance value
$C\dots$	Capacitance value

Sometimes the first-order filter circuits may not be enough to remove all unwanted signals, therefore in this work a second-order low pass filter is applied. The cut-off frequency of the first stage and the second stage is 16.9 Hz and 33.9 Hz respectively.

The reason for this conversion is that the MicroMod cannot accept a high-frequency input. And after the second-order filter, the current is close to a DC and approximately proportional to the output current from HVCU.

In the next step, the voltage signal after the filter is digitized with 16 bits and sent to MC in the PC via a PCAN-USB. After some calculation in MC, the calculated pressure sensor's output is sent to one MicroMod again in a CAN message and converted into an analog output.

In the PC, the data transmission between MC, Simulink model and PEAK device can be found in Figure 72. Via MC interfaces, the voltage after the filter is read as a 16-bit value in the Simulink model, and the corresponding control current value through the solenoid is determined by a lookup table (Bit2_Isoln_C0). The estimated hydraulic pressure is calculated from the control current based on a characteristic look-up table. Next, the voltage output of the pressure sensor can be obtained.

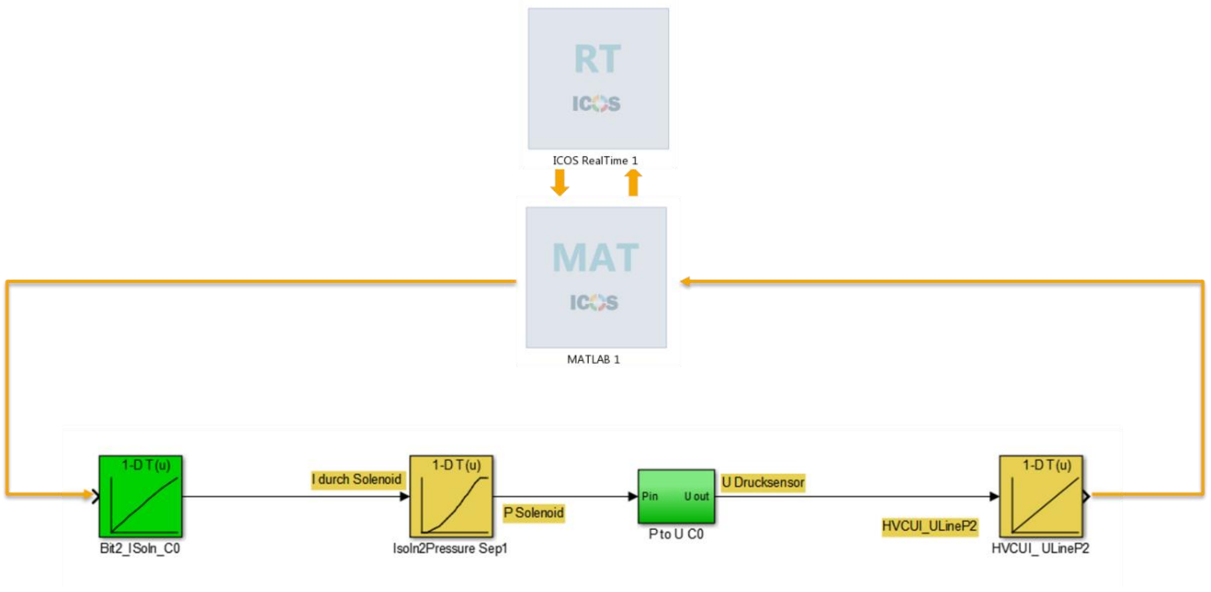


Figure 72: Data transmission between the Simulink model and ICOS interfaces

The output voltage of the pressure sensor is not only dependent on the pressure, but also on the supply voltage. It is calculated according to formula (6.2). The supply voltage is 5 volts here.

$$U_{sensor} = (k \cdot p - b) \cdot U_{supply} \tag{6.2}$$

U_{sensor} …	Output voltage of the pressure sensor [V]
k …	0.04379 [V/bar]
b …	0.005528
p …	Hydraulic pressure [bar]
U_{supply} …	Supply voltage of the sensor [V]

At the last stage in Simulink, it is necessary to convert the output voltage of the pressure sensor into a 12-bit value, in order to send it subsequently via MC and PCAN-USB to MicroMod Analog 2. Subsequently, the MicroMod outputs a voltage value from 0 to 5 volts to the HVCU. Finally, the voltage signal is obtained by the HVCU after a voltage follower circuit (Figure 73). The usage of voltage follower is because that a voltage follower has high input impedance and low output impedance, which can resolve the problem of loading effect and impedance mismatch. The voltage follower circuit here separates the MicroMod from the HVCU in case that the HVCU input influences the output of the MicroMod.

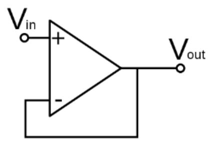


Figure 73: Voltage follower

After the correct configuration of the clutch separation control and lubrication control, the HVCU can close and separate the clutch properly.

6.4.2 Simulation of accelerator pedal sensors

In the plant model, the accelerator pedal position defined in the STIMULI (Signal Generator) passes through CRUISE and is converted into two voltage values of two accelerator pedal sensors (Figure 74).

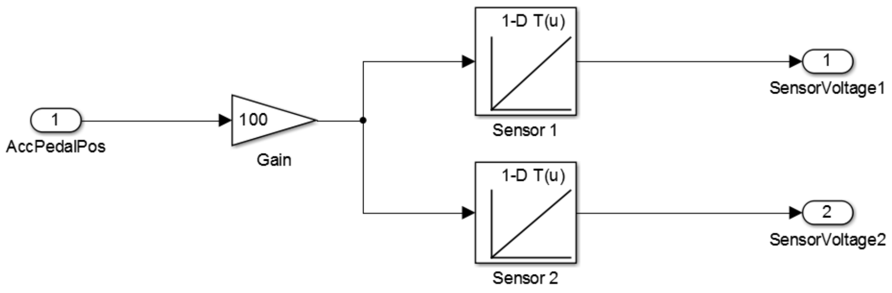


Figure 74: Converting accelerator pedal position to sensor voltage

The conversion formula is based on (6.3) and (6.4)

$$\mathrm{V}_{\mathrm{senor1}} = k_1 \cdot \mathrm{X} + b_1 \tag{6.3}$$

$$\mathrm{V}_{\mathrm{senor2}} = k_2 \cdot \mathrm{X} + b_2 \tag{6.4}$$

where $\mathrm{V}_{\mathrm{senor1}}$ and $\mathrm{V}_{\mathrm{senor2}}$ are the voltage output from two simulated sensors [V] and X is the position of accelerator pedal (from 0 to 100); $k_1 = 0.034$ [V], $b_1 = 0.8$ [V]; $k_2 = 0.017$ [V], $b_2 = 0.4$ [V].

In the ICOS_To_System block, according to a look-up table, the voltage signal is firstly restored to the original signal, the position of accelerator pedal (0 to 100), as shown in Figure 75. As the linearity of the output of MicroMod is not satisfying, the relationship between the digital input to the MicroMod and the analog output from the MicroMod, which is read by the HVCU, is measured and stored in the calibrated look-up table in the middle.

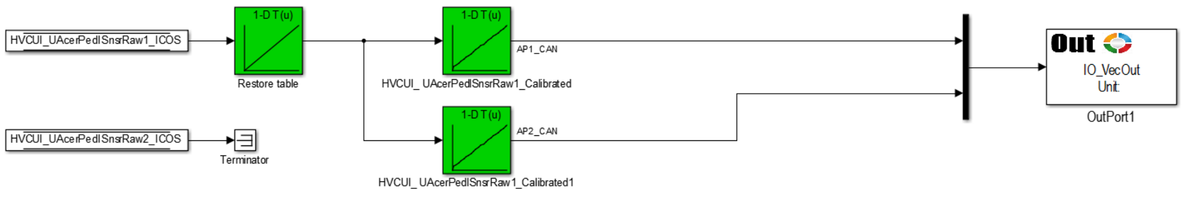

HVCUI_UAcerPedlSnsrRaw1_ICOS
1-D T(u)
Restore table
1-D T(u)
AP1_CAN
HVCUI_ UAcerPedlSnsrRaw1_Calibrated
1-D T(u)
AP2_CAN
HVCUI_ UAcerPedlSnsrRaw1_Calibrated1
HVCUI_UAcerPedlSnsrRaw2_ICOS
Terminator
Out
IO_VecOut
Unit:
OutPort1

Figure 75: Converting sensor voltage to 12-bit digital value

Similarly, a voltage follower is installed between analog outputs of the MicroMod and corresponding inputs of the HVCU to mitigate ripples on voltage signals.

6.5 Improvements to the real-time performance

Real-time factor that indicates the real-time performance is introduced first. The definition of the real-time factor is the time needed to calculate the current simulation step (including platform work) to the simulation time [37].

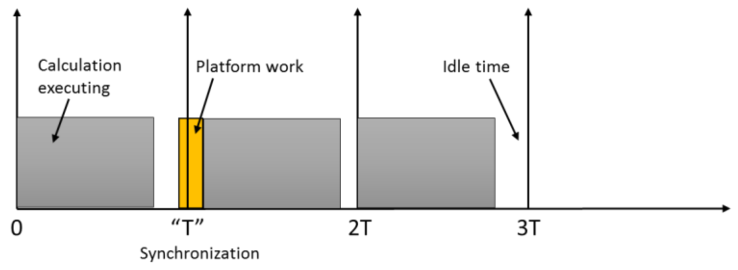


Figure 76: Real-time control timing

Figure 76 illustrates the real-time control timing and the mechanism to fill the hard real-time conditions.

If the synchronization time (macro time step) of the wrapper (like RealTime wrapper) in Model.CONNECT is T, the wrapper is operated at distinct points in time that are one time step apart. If the calculation of the offline system in the framework of MC takes more time than the synchronization time interval, an overrun situation takes place and the co-simulation cannot run in real-time. Moreover, whether every synchronization time "T" is really T is a big question, due to the fact that MS Windows is not a RTOS but a soft real-time OS. Also, in certain conditions, the time cost of the platform work as well as dead-time during data transmission cannot be neglected. In this work, the HVCU real test system has to meet the hard real-time conditions, since an involved component is a real and complex control unit.

Generally, there are two aspects of real-time co-simulation platform can be optimized: Making the calculation faster than real-time and reducing the jitter of synchronization. The platform work or the time of data transmission is not under control of the user and limited by software or hardware.

Considering the calculation time, one prerequisite is that the plant model under the MATLAB wrapper must be executed faster than synchronization time - so it has time slots for frame work of MC. For the same reason, the task of a PEAK device has to be finished within the defined time step size. Considering the synchronization time, the targeted real time factor of each Wrapper in the MC environment is to stabilize at "1" as long as possible. Thus the synchronization time of MC is predictable and MC can fulfil the hard real-time requirement as much as possible.

In the following parts, the efforts concerning guarantee of timely calculation of subsystems and mitigation of jitter of synchronization time are outlined.

6.5.1 Calculation time of subsystems

Application of vectors for data transfer

In the previous work, each signal was transmitted by one ICOS interface. Since one ICOS interface (InPort or OutPort) actually is an S-function and there are more than 400 ICOS interfaces in a Simulink model, the computation cost of the model was markedly elevated. Moreover, one additional MATLAB wrapper was needed in the MC environment, which separates vectors into single signals and combines single signals into vectors.

To resolve this problem, vector data transfer is put to use, which means that one ICOS interface can transmit hundreds of signals in the format of a vector. This is defined in the dimension of each ICOS InPort or OutPort block.

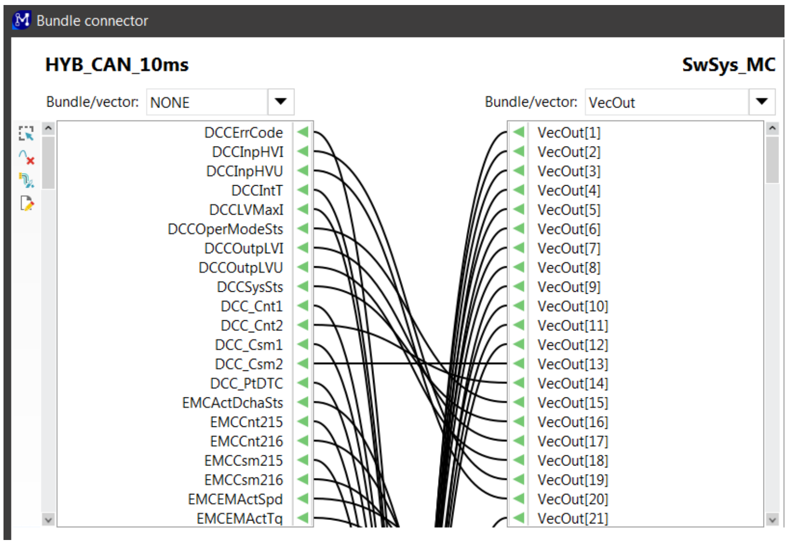


Figure 77: Vector connected with CAN signals in MC

The flaw of the ICOS vector is that instead of a specific signal name, each signal is named as Vector [number] (Figure 77), thus it is impossible to connect them manually or with the help of connection suggestions in MC. Several MATLAB functions are created so as to generate a *.CSV file, which can be imported into MC and create lines between components automatically. The format of a *.CSV is as follows:

\ SwSys_MC, , icosVecOut, 1, ~, \ PT_CAN_100ms, , ACCycMod ,

\ SwSys_MC, , icosVecOut, 2, ~, \ IO_CAN, , AP1_CAN ,

\ SwSys_MC, , icosVecOut, 3, ~, \ IO_CAN, , CCSwitch_CAN ,

Finally, the essential improvement with the application of vector data transfer can be viewed in Figure 78.

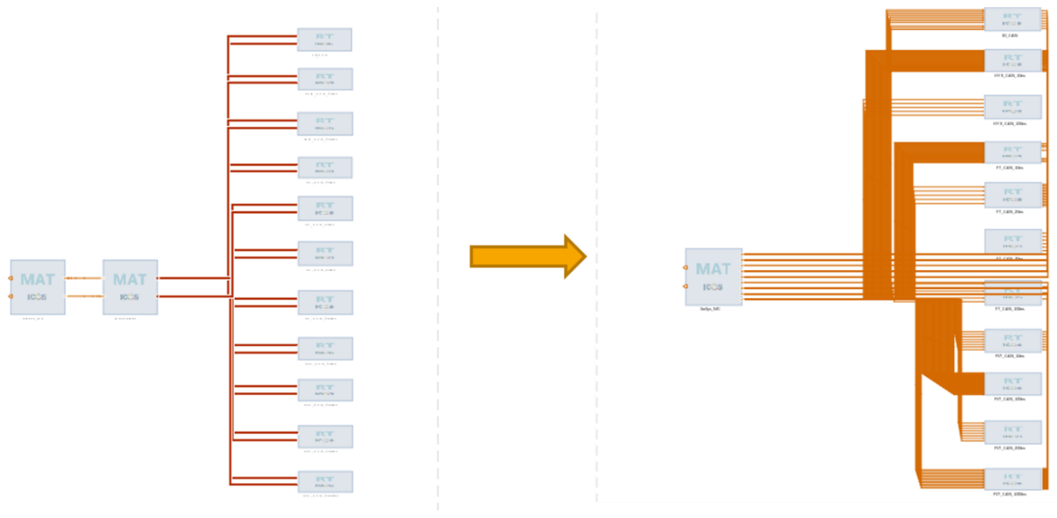
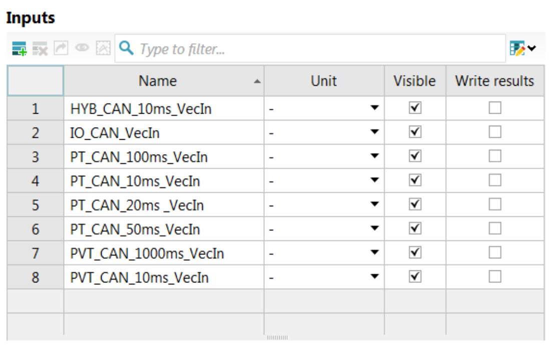


Figure 78: Improvement to MC structure with the application of vectors

In Figure 78, the right structure does not need the middle block for muxing and demuxing anymore, in comparison with the left one. Two output vectors from MATLAB and six input vectors to MATLAB implement the data transfer instead of nearly 400 single signal interfaces. The employment of vectors for transferring data can greatly reduce the computation efforts of the Simulink model, discard the extra MATLAB wrapper and therefore strengthen the real-time ability.

Deactivating saving data

In a Simulink model, blocks of saving data or functions of data logging are not acceptable since this remarkably extends the computation time. Moreover, in the MC it is essential to deactivate the Write results function, which saves the data across ICOS interfaces.



Inputs

Type to filter...

	Name	Unit	Visible	Write results
1	HYB_CAN_10ms_VecIn	-	✓	☐
2	IO_CAN_VecIn	-	✓	☐
3	PT_CAN_100ms_VecIn	-	✓	☐
4	PT_CAN_10ms_VecIn	-	✓	☐
5	PT_CAN_20ms _VecIn	-	✓	☐
6	PT_CAN_50ms_VecIn	-	✓	☐
7	PVT_CAN_1000ms_VecIn	-	✓	☐
8	PVT_CAN_10ms_VecIn	-	✓	☐

Figure 79: Deactivating the Write results in MC

Accelerator mode in Simulink

A further computation time saving is obtained by changing the compiler settings from Normal to Accelerator in Simulink. The Accelerator mode replaces the interpreted code normally used in Simulink simulations, therefore shortening the model run time [44]. The disadvantage, however, is that the model needs longer time to compile.

Initialization time

One feature of MC is that in the initial several time steps, the real-time factor is significantly higher (possibly more than 3000). This is because each block needs time for initialization. If this time cost is counted in the overall co-simulation time, then it is found out that the co-simulation is much longer than the defined simulation time. The solution for this is to provide an initialization period at the beginning when only default values are sent from MC, which enables MC to catch up the real-time device.

Attention to warning information

As Windows OS is not a RTOS and many factors could make the calculation of a model slower, even though normally it is fast enough. It is important to check the information of the log file as well as timing output after each test.

If the calculation is slower than real-time then there will be a warning in the log file, like:

Warning: SwSys_MC: ! Target-time 20.48 s: Time-step 0.01 s: calculation took 0.010284045093442273 s !

In this case the test result is not trustable and should be neglected.

6.5.2 Jitter of synchronization time

Application on a workstation with better configuration

For the purpose of obtaining a better real-time performance, the PC configuration is one of the most crucial factors. It is obvious that a more powerful PC is beneficial to shortening the calculation time. Furthermore, it can relieve the problem of jitter from Windows OS. For example, the RealTime wrapper of MC is determined to exchange the data at a fixed time step (e.g. 10 ms), however, if a high-priority process (like a system process) is implemented near the time point, the process of RealTime Wrapper has to wait and a time delay of exchange of data unavoidably takes place. And if excessive work load is exerted on the processor, the OS might become less responsive. This makes it hard to respond in accordance with the wall clock time. A workstation with superior configuration is able to deal with these problems more excellently.

To gauge the promotion brought by the PC configuration, a same test case was implemented on a laptop (4 cores, i5-5300U CPU @ 2.30GHz) and on a desktop (32 cores, CPU E5-2640 v3 @ 2.60GHz). The comparison of the real time factor of these two cases is shown in Figure 80.

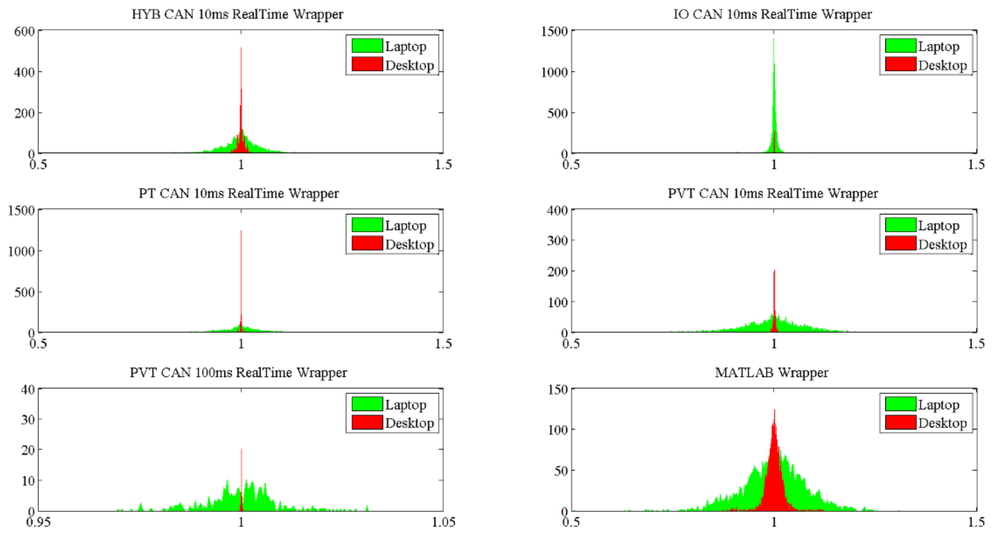


Figure 80: Comparison of the real time factor on a laptop and a desktop

Figure 80 is created by histogram plot function in MATLAB. In each sub-figure, the histogram reveals the underlying shape of the distribution of the real-time factor of one wrapper during the entire co-simulation (except initial 5 steps). In the figure, the y-axis indicates the number of time steps while the x-axis denotes the real-time factor.

It is evident that the distribution of the real-time factor of the simulation on the desktop is much closer to “1”. Thus the possibility of missing the deadline on the desktop is considerably lower but still cannot be fully eliminated.

The optimal workstation would be a desktop whose CPU clock is more than 3.3 GHz (the number of cores helps a little).

Deactivating online monitoring

Online monitoring of MC could affect simulation speed. If this function is not necessary during simulation, online monitoring is suggested to be deactivated.

A clean test environment of workstation

In Windows OS, it is important to minimize the number of processes, which might block the operation of MC. During the co-simulation, all unnecessary processes should be shut down and it is better not to use the workstation (like moving the mouse). The remote desktop connection with a server is not suggested since it might lead to a cyclical loss of CAN signals.

There is one possibility that the co-simulation-related processes can be set to high priority in Windows Task Manger. The influence of this kind of setting is described in chapter 6.9.1.

Mitigation of jitter of communication with external I/O devices

Previously, all PCAN-USBs were connected with an unpowered USB hub, which is probably not capable of supporting the PCAN-USBs. A powered USB hub is now utilized, which can provide a stable power supply for PCAN-USBs and enhance their stability.

Using better devices can also help. In general, Vector CAN adapters are more reliable than PCAN-USBs. And PCI (Peripheral Component Interconnect) is more compatible with real-time systems in comparison with USB (Universal Serial Bus).

6.6 Update of software in the HVCU

6.6.1 Workflow

With one year passing, the software in the HVCU, which was tested, has been modified considerably. Thus it is necessary to generate new ICOS interfaces, build up a new project from scratch and also create a more readable and clear environment.

The workflow of this task can be shown in Figure 81.

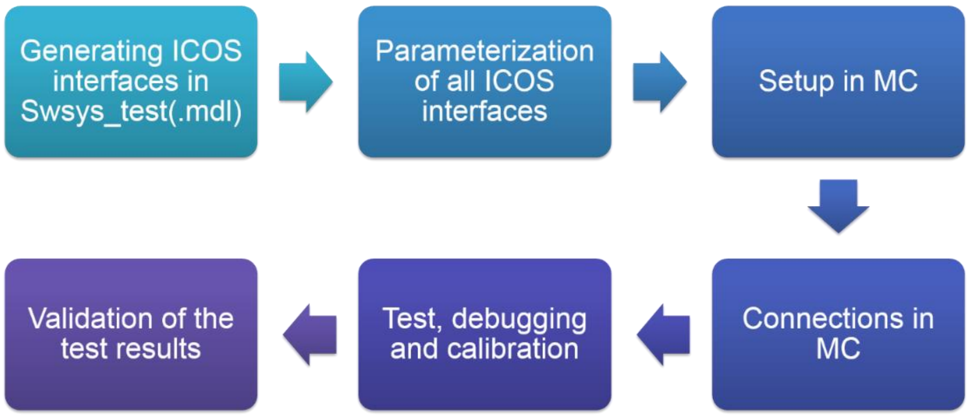

Generating ICOS interfaces in Swsys_test(.mdl)
Parameterization of all ICOS interfaces
Setup in MC
Connections in MC
Test, debugging and calibration
Validation of the test results

Figure 81: Workflow of HVCU real test

The first two steps are generation of ICOS interfaces and parametrization. Next, the setup in MC is similar to the previous MC project. After this, the connection of ports in MC environment is implemented by importing a *.CSV file generated by MATLAB functions. The following step is to test and debug the test system. Meanwhile since the plant model is different from a real vehicle, additional calibration of the HVCU is necessary. Last but not least, validation of the test results must be carried out.

6.6.2 ICOS_To_System block

Similar to the Multi-ECU virtual test, a supplementary block (ICOS_To_System block) for data exchange is a necessity in the AVLab environment. Inside ICOS_To_System block (Figure 82), there are three blocks; ICOS_VecIn, Calculation and ICOS_VecOut.

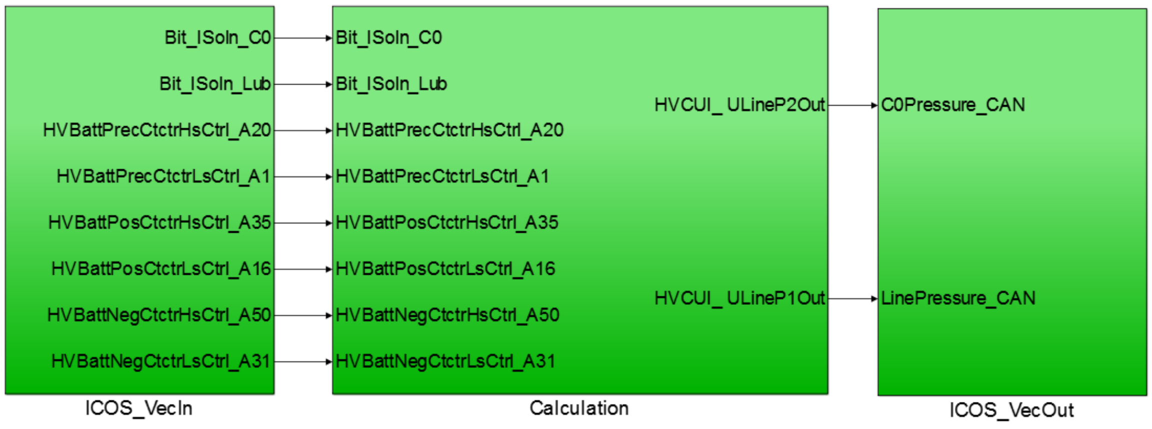


Figure 82: ICOS_To_System block

In ICOS_VecIn block, all signals from the HVCU or MicroMods are obtained by ICOS InPort blocks, and then they are either connected with DataStore blocks for further usage in the plant model, terminated or sent to the Calculation block (Figure 83).

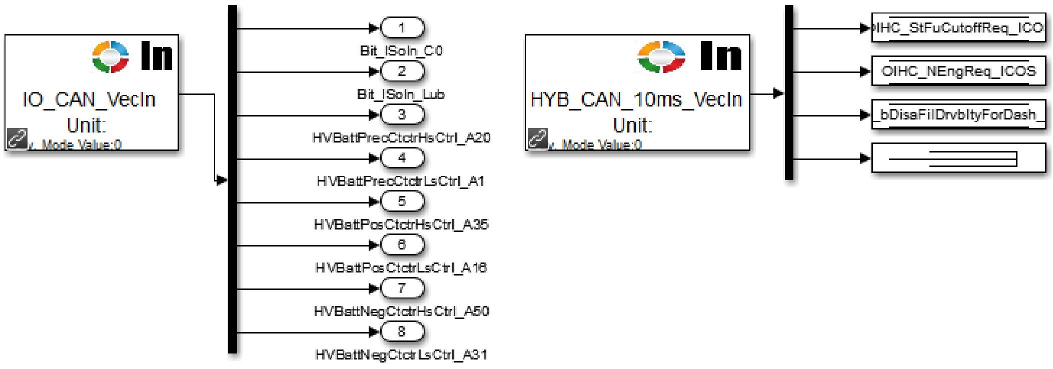


Figure 83: ICOS_VectorIn block

In Calculation block, some inputs are calculated here and then the results are sent to ICOS_VecOut block.

In ICOS_VecOut block, the signals from the plant model are sent to the output vector through DataStore blocks. For those output signals, which are not from the plant model but are indispensable for the HVCU, a constant or a simple calculation is given according to their physical meanings (Figure 84).

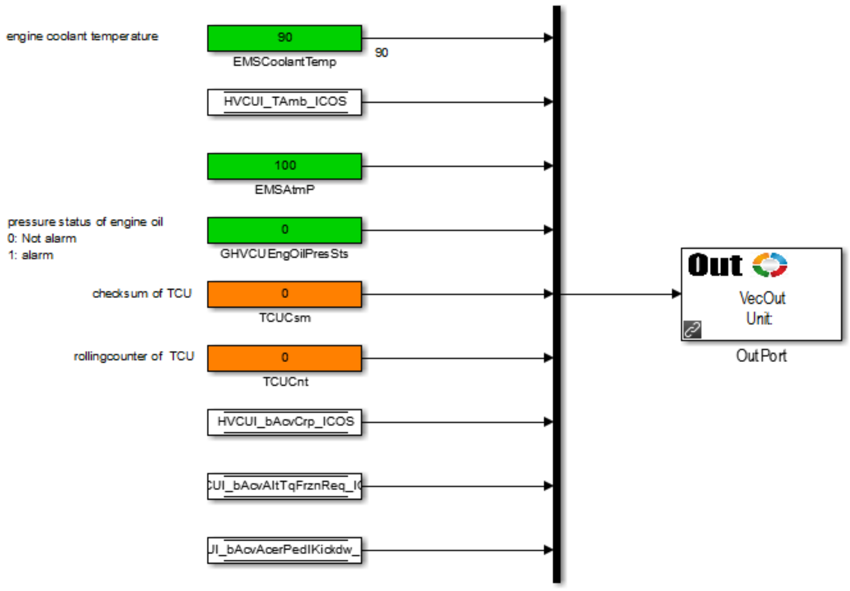


Figure 84: ICOS_VecOut block

6.6.3 Automatic generation of interfaces and parameterization

The number of inputs and outputs of the latest software has been doubled compared with the old software, which means that a manual generation of interfaces, renaming and manual connection of lines would be impossible. Some scripts were written in order to fulfil the task of automatic generation of interfaces and parametrization of them.

By using the scripts, the work load is reduced and manual errors can be avoided. Moreover, it is easy to create a more readable structure (different type of signals have different colors and size of each block as well as the distance between each block is under control), which is beneficial to further inspections.

6.6.4 Some other improvements

As described in chapter 6.1.1, the HVCU contains the C0 clutch control software. One big challenge is that the plant model also has a simple C0 control model. The simple C0 control model cannot be simply deleted because its outputs are inputs to the CRUISE. This conflict might lead to some unexpected behavior of the clutch. If the signals given to the C0 control software in the HVCU are the same as those given to the simple C0 control model in the plant model, the influence of this conflict can be highly reduced since they use the almost same methods to calculate the control strategy of the clutch.

Previously, OIHC_TqCluSepReq (Torque Clutch Separation Request), which is an input for the C0 control, is an internal signal in the HVCU and cannot be read via CAN bus. Thus, a CAN signal OIHC_TqEngReq (Torque Engine Request) replaced it when the HVCUI_StEng (State of Engine) was "1" (Figure 85). But this did not reflect the real situation in the vehicle.

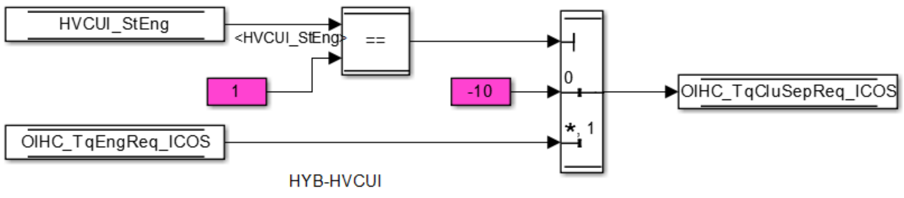


Figure 85: Previous signals for Torque Clutch Separation Request

Fortunately, in the updated software some internal signals of the HVCU are sent out via CAN bus for debugging. Therefore, correct signal DBG_OIHC_TqCluSepReq (Debugging signal of Torque Clutch Separation Request) can be given to the simple C0 clutch control model in the plant model.

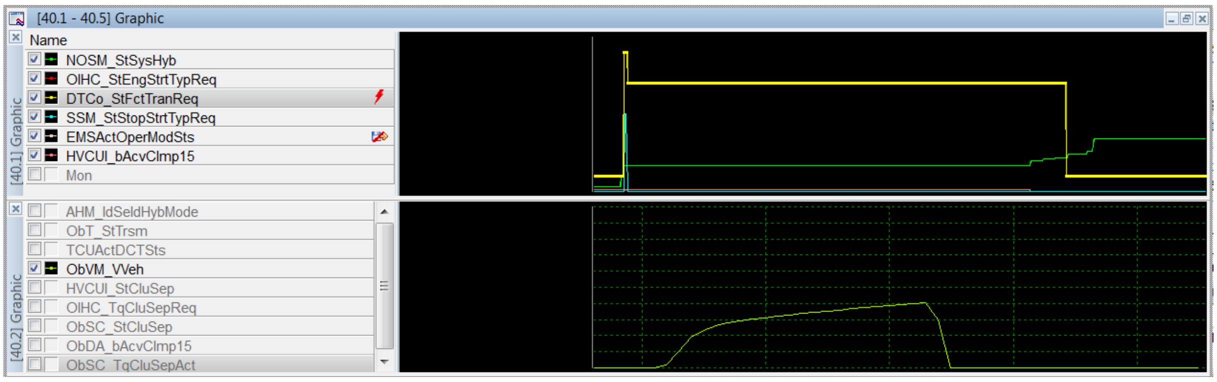
6.7 Automatic Testing

The intention of the automatic testing is that once the preparatory jobs are finished and the automatic testing is started, a batch of test cases can be tested sequentially and their results can be saved without additional manual work. This means that even no worker sits in front of the screen, the testing can run without problems and the efficiency of testing is therefore significantly enhanced.

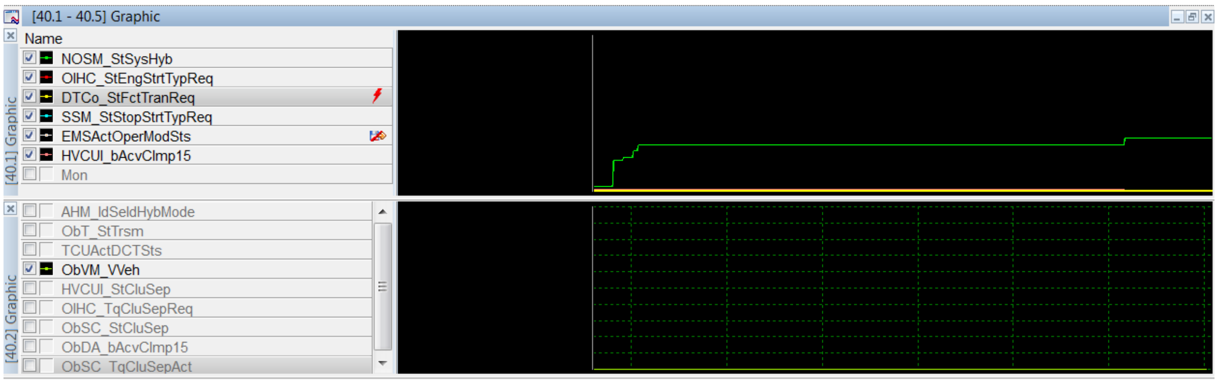
6.7.1 Resetting of the status

For the objective of automatic testing, an overriding challenge is that the control unit and supplementary hardware has to restore to the initial state after implementation of a test, otherwise another new test case is prone to fail. For example, after one test case, the signal NOSM_StHybSys, which means the state of the hybrid system, is SHUT DOWN. In this case when another test case is started, the state of the HVCU remains SHUT DOWN and the new test case is not possible to run as defined. The only solution is to switch off the HVCU after one test case and switch it on again, which is able to set all signals and states of the HVCU to default values, before the new test case begins.

In the first try, The Clamp 15 is employed for the implementation of resetting of the status. The Plant model sends out a binary CAN signal (Clamp 15, an ignition signal) via IO CAN bus to one MicroMod. If Clamp 15 is "0", the MicroMod outputs 0 volts captured by the HVCU and after nearly 1.5 seconds the HVCU will be power-off due to a time delay relay component. If Clamp 15 is "1", the MicroMod outputs 10 volts, and the HVCU is switched on immediately. However, the resetting of the HVCU by Clamp 15 cannot always work. The result of a successful test and the result of the following test using Clamp 15 to reset the HVCU are displayed in Figure 86. Both tests are implemented with the same test case.



(a) A successful test result



(b) The following test result using clamp 15 to reset HVCU

Figure 86: Failure of using Clamp 15 to reset the HVCU

The most possible reason for this is that MicroMods hold last values, which means that MicroMod remains the output if there is no input available. At the initialization stage of the test the control unit still receives "old" values from the previous test case rather than default values (usually 0), which might lead to an abnormal behavior.

Nevertheless, the simplest solution to this problem is that all MicroMods are also switched off after finishing one test case and then switched on. Therefore, after one test case all MicroMods are reset and send out default values defined in MicroMod configurations.

To achieve this goal, one time delay relay whose delay time range is from 0.5 to 5 seconds is utilized here and the delay operates in the direction of closing. The circuit diagram and function diagram of this electronic component can be found in Figure 87. In this thesis, delay time is set to 5 seconds.

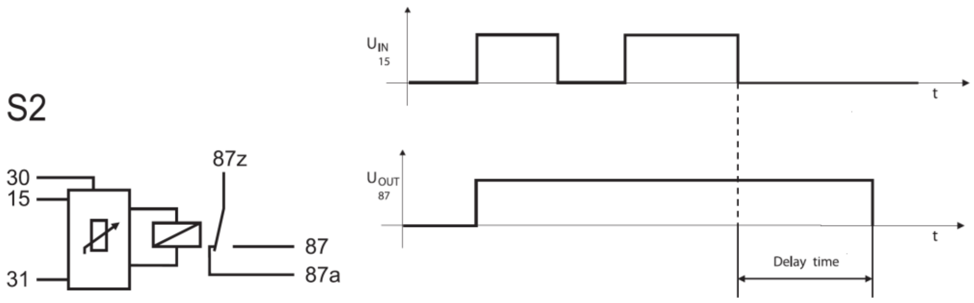


Figure 87: Circuit diagram and function diagram of the time delay relay [45]

In addition, a new ICOS interface (Power Switch) is also appended in the plant model, as shown in Figure 88. If the simulation time excesses a defined time limit, then the Power Switch will be “1”, and the power supply of all hardware will be cut.

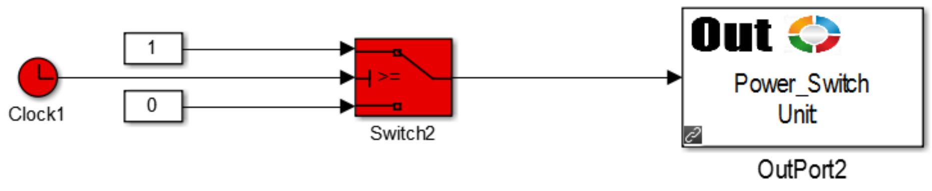


Figure 88: A new ICOS interface for Power Switch in the plant model

A simplified circuit of the power switch system can be viewed in Figure 89.

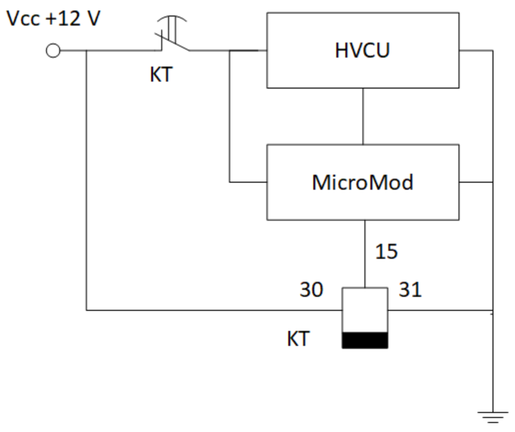


Figure 89: A simplified circuit of the power switch system

Initially, the MicroMod sends out Power Switch signal with a default value “0” and the contact keeps touching. Once the Power Switch value is “1”, the whole system’s power is cut off by the relay. Following this, MicroMod is not capable of sending out any voltage, which means that Power Switch value is 0 again, so that after 5 seconds the contact touches again and the power supply of the whole system recovers. Thus all devices are set to their initial states and are ready for an upcoming new test case.

6.7.2 Scripts in CANape

CANape provides powerful function libraries consisting of different kinds of function groups, such as Measurement Functions, Device Functions, or Script Functions and so on. The functions are defined in a C-like programming language and some of them are real-time and support implementation during the measurement, like Stop( ), IsRuning( ), etc.

One script is created in CANape for automatic calibration of some parameters before the measurement starts. Another script is created for automatically setting the device online/offline and starting/stopping the measurement in combination with a trigger signal from HVCU and a trigger signal controlled by the plant model. The block diagram of the script, which is used for automatic testing is shown in Figure 90.

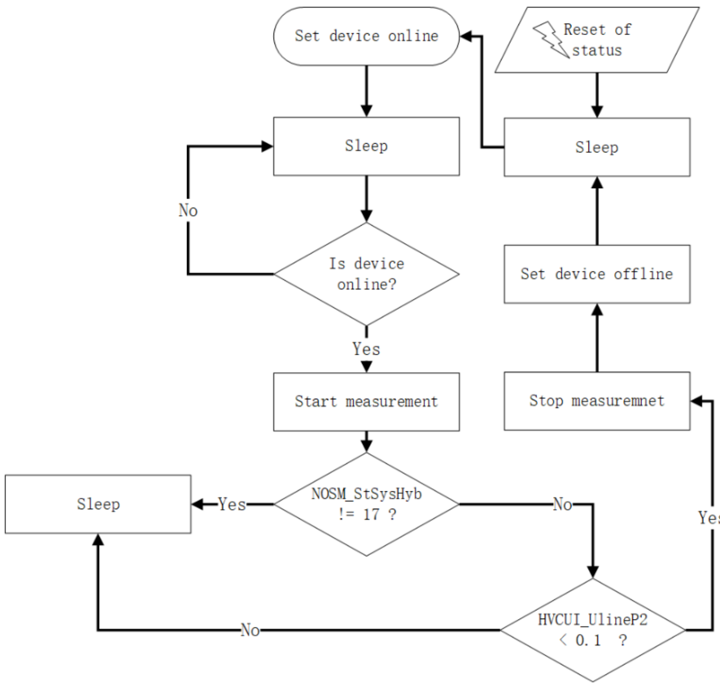


Figure 90: Block diagram of the script used for automatic testing

6.7.3 Existing problems of the automatic testing

There are still some problems existing in the topic of automatic testing. The first problem is that sometimes the warnings from other programs (TargetLink, Integrity, AVLab, etc.) could block the ongoing MC. In this case, it is requisite to click OK on those warning dialogues for proceeding. These warnings might be simply due to some features or bugs of the related programs.

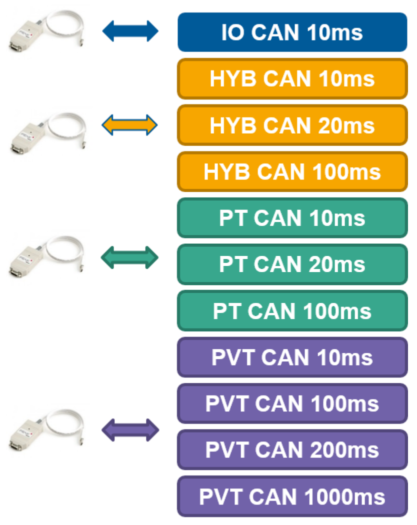
Also, before the co-simulation starts, during the measurement sometimes the device suddenly becomes offline no matter if the script is utilized or not. This leads to an error of measurement. Actually the control unit has been already connected to CANape and set online. It seems that sometimes the HVCU shuts down itself for resetting during an error state. In this case, a manual intervention is inevitable. To resolve this problem, setting device online and starting measurement shall be precisely a few seconds before the co-simulation really starts in order to avoid confrontation with resetting of HVCU during the CANape measurement.

6.8 Test specification

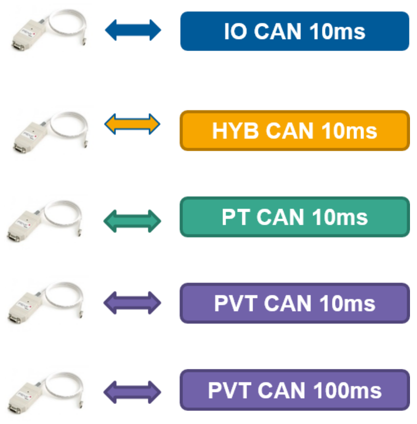
6.8.1 A solution to the instability of MC with PCAN-USBs

Until Model.CONNECT R2017f Build 091, an overriding challenge of the HVCU real test is the instability of MC with PCAN-USBs. The success rate of the test using RealTime wrappers with PCAN-USBs was approximately only 10%, thus it took more than one hour to obtain one satisfying test result.

The reason why this happened has been detected with the assistance of the PEAK support team. The previous connection between RealTime wrappers and PCAN-USBs is displayed in Figure 91(a), in which several RealTime Wrappers with different frequency are connected with one PCAN-USB. Due to the mechanism of RealTime wrappers, a crash frequently takes place when more than one RealTime wrappers (e.g. HYB CAN 10ms and HYB CAN 20ms) attempt to access one PCAN-USB.



(a) Previous connections



(b) New connections

Figure 91: Previous and new connection between RealTime wrappers and PCAN-USBs

So as to solve this problem, a workaround is designed, which can be shown in Figure 91(b). In HYB CAN RealTime wrappers of the MC, CAN messages of HYB CAN which were sent or received every 20 ms and 100 ms are now sent or received every 10 ms. The similar change happens in RealTime wrappers of PT CAN. For PVT CAN, the original four RealTime wrappers are merged into two RealTime Wrappers whose time step is 10 ms and 100 ms respectively.

The drawbacks of the workaround are multiple. Firstly, the work load of MC is considerably increased along with higher frequency of sending and receiving signals. But this brings no benefit because the sampling frequency of a RealTime wrapper only needs to be as same as the defined frequency on CAN bus. Moreover, the bus load rises. For example, according to the estimation of the CANape, the bus load of PVT CAN increases approximately from 54% to 64%. Last but not least, the data transmission on CAN buses of the workaround is different from it on CAN buses in a real vehicle.

Despite the aforementioned disadvantages, the workaround is still applied. This is because with this workaround, the problem of instability of MC with PCAN-USBs never happens again and the further work can proceed. Also, the workaround has one benefit: The number of co-simulation processes markedly diminishes. This is because every RealTime Wrapper creates one runKernel.exe and conhost.exe during a co-simulation. The reduced number of processes might be helpful to mitigation of jitter of synchronization time.

Another feasible solution is the utilization of Vector CAN adapters or PCAN-PCI cards, which do not cause instability of MC.

6.8.2 Test arrangement

Figure 92 displays the test arrangement of HVCU real test system.

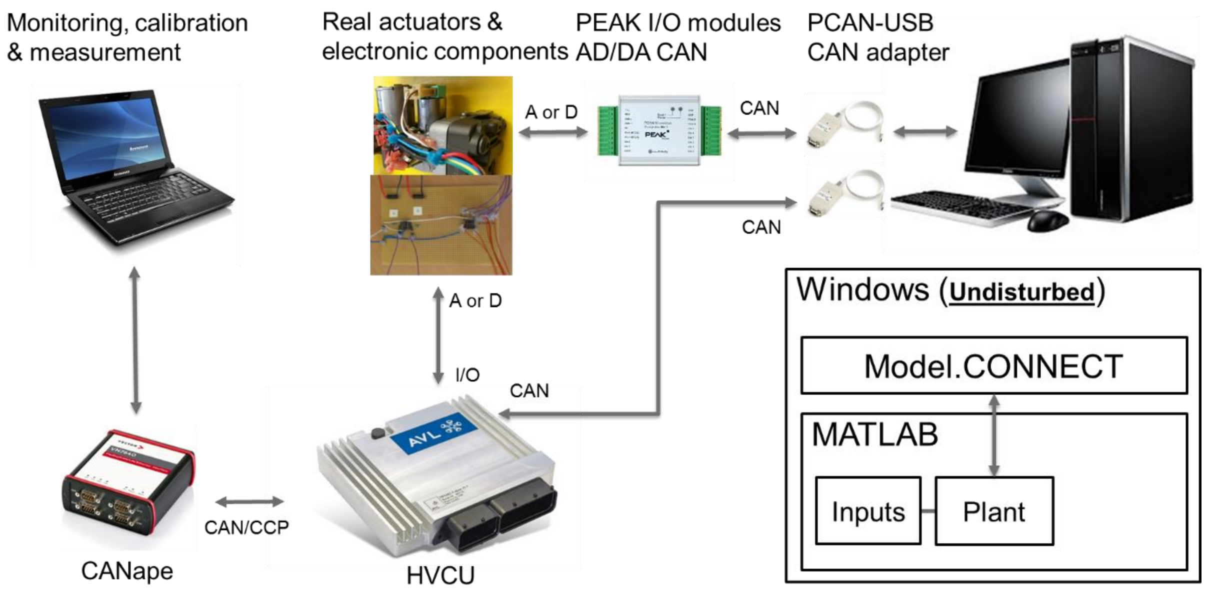


Figure 92: Test arrangement of HVCU real test

In the test arrangement, two computers are needed, one for execution of MC and another for CANape. Both computers have a Windows7 64-bit OS. As the co-simulation with the HVCU requires hard real-time conditions, MC as well as MATLAB operates on an undisturbed workstation (right one). The plant model with inputs is implemented in MATLAB. On the left PC, CANape can record all signals on three CAN buses. Also, it can record and calibrate the internal signals of the HVCU. During the measurement, all measured signals can be displayed in various windows of the left computer, so that the user can monitor the ongoing test. As MC provides an interface between real world and virtual world, the plant model can exchange data directly with HVCU via CAN bus, or via CAN bus and I/O modules in cooperation with real actuators as well as electronic components.

A more detailed test arrangement can be found in Figure 93.

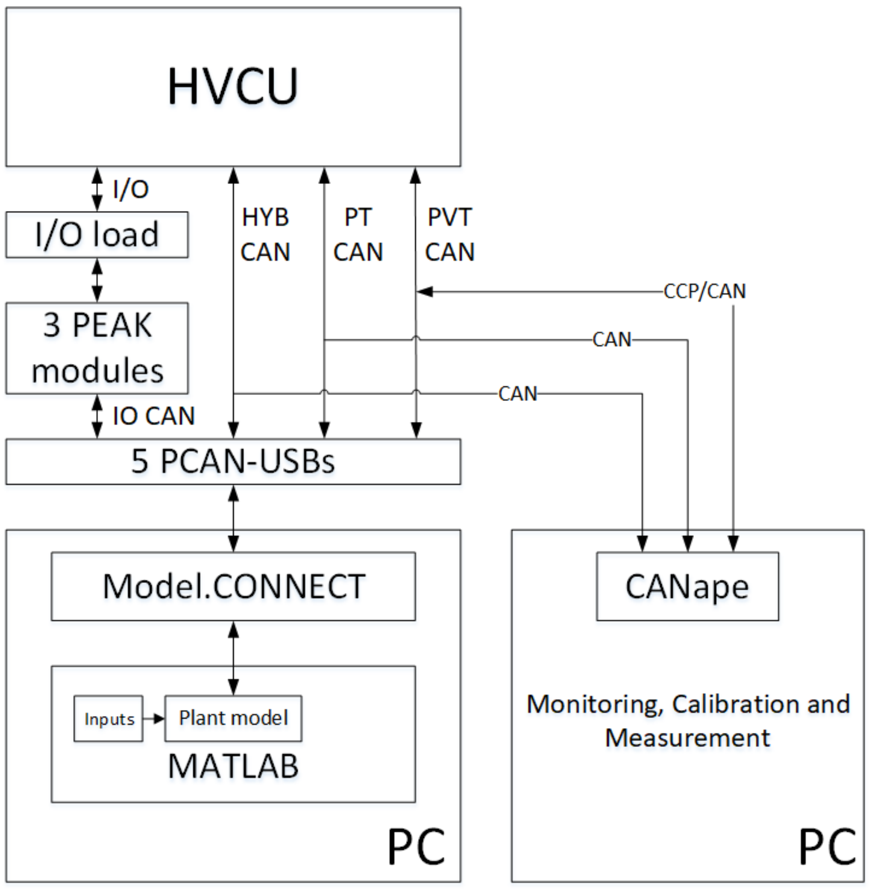
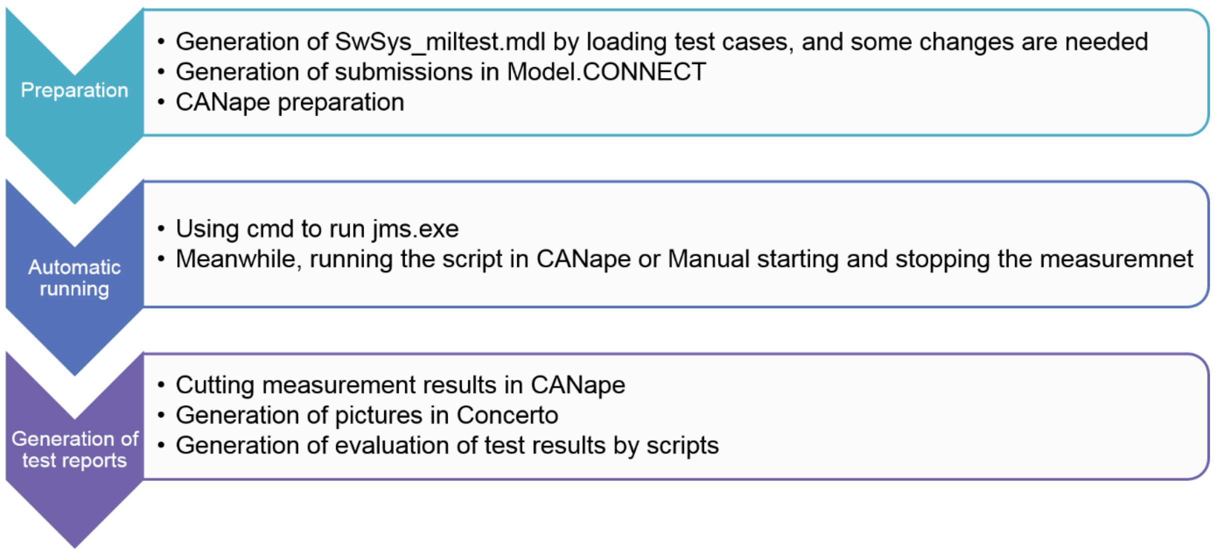


Figure 93: Detailed test arrangement

6.8.3 General test sequence

The general test sequence of HVCU real test is shown in Figure 94.



- Generation of SwSys_miltest.mdl by loading test cases, and some changes are needed
- Generation of submissions in Model.CONNECT
- CANape preparation

- Using cmd to run jms.exe
- Meanwhile, running the script in CANape or Manual starting and stopping the measuremnet

- Cutting measurement results in CANape
- Generation of pictures in Concerto
- Generation of evaluation of test results by scripts

Figure 94: General test sequence

At the beginning, different Simulink models with different inputs are prepared. Once the AVLab is available and the AVLab project is loaded, simply by clicking Load TC, a new test case is loaded (Figure 95). In addition, some other modifications are required in the model: the simulation time of the model, the threshold time for Power Switch for resetting of the whole test system, and the threshold time for trigger signals in case of automatic testing. In some cases, supplementation of a time range for initialization of MC in the signal builder is necessary.

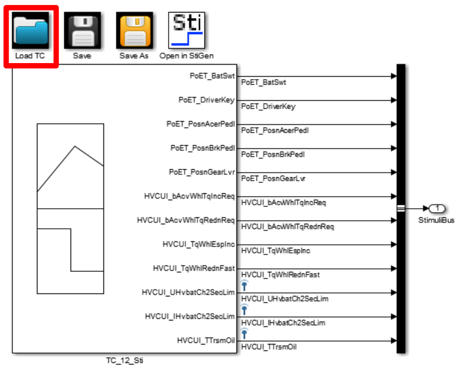
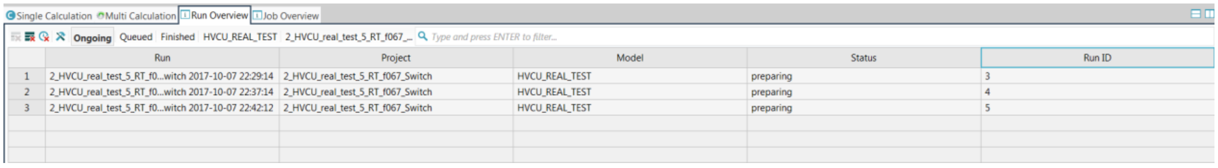


Figure 95: Loading a new test case

Once Simulink models are available, the submission of each Simulink model can be generated in MC. In order to use manual submissions, selected resource type in Computing Resource should be set to be manual firstly. For the objective of generating manual job submissions linked with different Simulink models, each prepared Simulink Models' path is loaded, after which its corresponding submission is initialized by clicking Run.

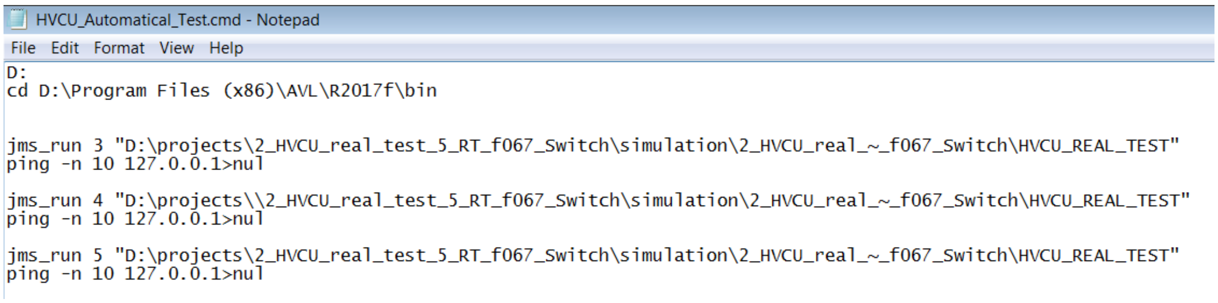
Finally, the manual job submissions are in status of preparing and their Run ID can be viewed in Figure 96



	Run	Project	Model	Status	Run ID
1	2_HVCU_real_test_5_RT_f0...witch 2017-10-07 22:29:14	2_HVCU_real_test_5_RT_f067_Switch	HVCU_REAL_TEST	preparing	3
2	2_HVCU_real_test_5_RT_f0...witch 2017-10-07 22:37:14	2_HVCU_real_test_5_RT_f067_Switch	HVCU_REAL_TEST	preparing	4
3	2_HVCU_real_test_5_RT_f0...witch 2017-10-07 22:42:12	2_HVCU_real_test_5_RT_f067_Switch	HVCU_REAL_TEST	preparing	5

Figure 96: Manual job submissions and their Run ID

Meanwhile, a new *.cmd (or *.dat) file is created to activate each manual job submission with the usage of Jms_run.exe (Figure 97).


HVCU_Automatical_Test.cmd - Notepad
File Edit Format View Help

```
D:
cd D:\Program Files (x86)\AVL\R2017f\bin

jms_run 3 "D:\projects\2_HVCU_real_test_5_RT_f067_Switch\simulation\2_HVCU_real_~_f067_Switch\HVCU_REAL_TEST"
ping -n 10 127.0.0.1>nul

jms_run 4 "D:\projects\\2_HVCU_real_test_5_RT_f067_Switch\simulation\2_HVCU_real_~_f067_Switch\HVCU_REAL_TEST"
ping -n 10 127.0.0.1>nul

jms_run 5 "D:\projects\2_HVCU_real_test_5_RT_f067_Switch\simulation\2_HVCU_real_~_f067_Switch\HVCU_REAL_TEST"
ping -n 10 127.0.0.1>nul
```

Figure 97: An example of *.cmd file for activating manual job submissions

On the other side, it is important to ensure that all hardware is properly set up and CANape's graphical user interface is available.

After the preparative phase, the serial co-simulation tests are started by running the *.cmd file. The CANape script for automatic testing can be used or simply the user manually sets the device online/offline and clicks start/stop of measurement in CANape.

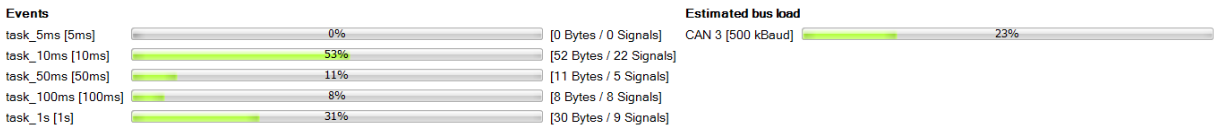
Once all pre-defined tests are accomplished, the valuable time range of test results can be cut out by CANape, then the test results in format of *.MDF can be further processed by Concerto, in which some sophisticated diagrams can be generated. Since in Concerto it is possible to define layouts, with the help of some scripts, automatic evaluation of test results and generation of test reports is possible.

6.8.4 Other suggestions for a successful test

CANape

CANape is a critical tool during the test, which calibrates internal parameters of the HVCU and records the CAN signals as well as internal signals. It should be noted that the increasing number of recorded internal signals leads to a higher bus load, which could block the start of the MC co-simulation.

To avoid this from happening, in principle the estimated bus load of CANape should be below 30% (Figure 98).

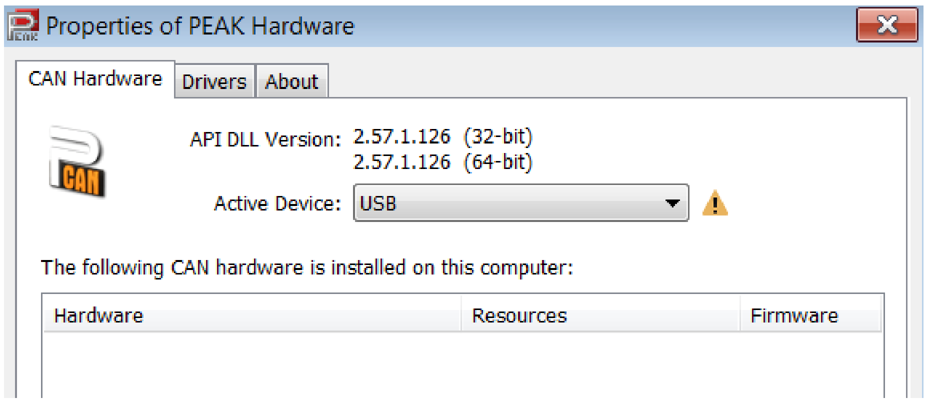


Events			Estimated bus load	
task_5ms [5ms]	0%	[0 Bytes / 0 Signals]	CAN 3 [500 kBaud]	23%
task_10ms [10ms]	53%	[52 Bytes / 22 Signals]		
task_50ms [50ms]	11%	[11 Bytes / 5 Signals]		
task_100ms [100ms]	8%	[8 Bytes / 8 Signals]		
task_1s [1s]	31%	[30 Bytes / 9 Signals]		

Figure 98: Estimated bus load in CANape

PEAK module

After connecting the PCAN-USB to the PC, it is necessary to check the properties of the PEAK hardware and ensure that the active device is USB (Figure 99).



Properties of PEAK Hardware

CAN Hardware | Drivers | About

API DLL Version: 2.57.1.126 (32-bit)
2.57.1.126 (64-bit)

Active Device: USB

The following CAN hardware is installed on this computer:

Hardware	Resources	Firmware

Figure 99: Properties of PEAK Hardware

And as described in chapter 3.4.3, each PCAN-USB device’s handle must correspond to the channel defined in the * .ini file, which means that they should be connected with the PC in a certain sequence.

6.9 Evaluation of the proposed test system

Considering the general requirements for HiL test systems, reliability of the test results and further practical usage, seven criteria are built up for investigations and assessment of the HVCU real test system. The seven criteria are:

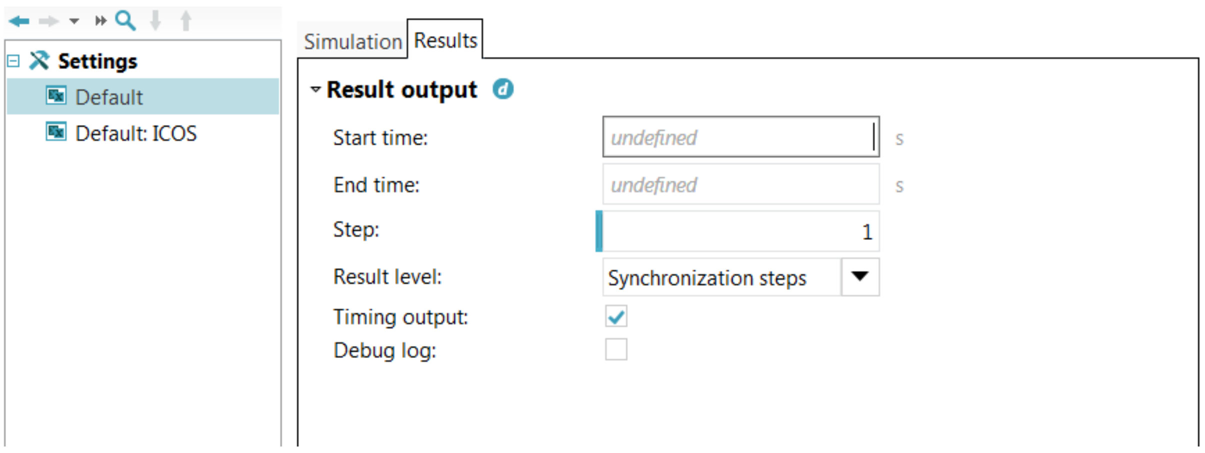
- Real-time performance
- Reliability (the test system can reflect the behavior of a real car)
- Repeatability of test results
- Applicability for various test cases
- Ease of setup for further projects or different ECUs
- Ease of testing
- Possibilities for extension

The detailed evaluation according to these criteria is elaborated as follows.

6.9.1 Real-time performance investigations

Investigations of 150-second test case

In a real-time co-simulation, the hard real-time condition of the test system is extremely important especially when it comprises an ECU. In this thesis, two parameters are proposed to gauge and quantify the real-time performance. One is the proportion of the real-time factor of calculation more than "1" and another is 3σ (three standard deviations) of the real-time factor. By applying these, the real-time performance of the test system can be easily evaluated. A test case whose simulation time is 150 seconds is created and used to give an insight into the real-time performance of the co-simulation platform. By activation of timing output (Figure 100), the timing information of the co-simulation is reordered.



Start time: undefined s
End time: undefined s
Step: 1
Result level: Synchronization steps
Timing output: ✓
Debug log:

Figure 100: Activation of Timing output

The real-time factor of calculation is defined as the calculation time of the wrapped element (like MATLAB) to the simulation time step. More specifically, it is the time period from the time that the calculation starts and to the time that the calculation results are available. In principle, it shall be always less than "1" except several initial time steps, otherwise an unavoidable time delay of synchronization takes place. The proportion of the real-time factor of calculation for different wrappers is shown in Figure 101.

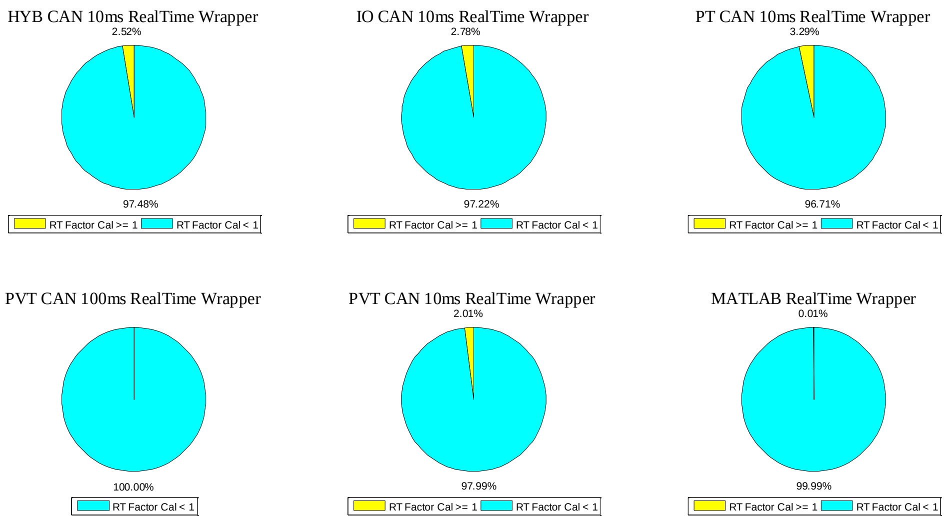


Figure 101: Proportion of the real time factor of calculation for different wrappers

In Figure 101, all calculation of PVT CAN 100ms can be finished within 100 ms. The performance of MATLAB is also satisfying. Only nearly 0.01% of real-time factor of calculation for MATLAB is more than “1”. This means that only one or two steps of calculation out of 15000 steps is longer than 10 ms (defined time step size in Simulink). For wrappers like PT CAN 10ms, HYB CAN 10 ms, a small share of real-time factors of calculation outnumber “1”. A future target is to reduce it as small as possible and ideally make it “0”.

One possible reason for this is that high-frequency and a large amount of calculation presents a challenge to the PEAK programs under these wrappers. A better distribution of work load for these wrappers needs to be considered. As discussed in chapter 6.8.1, returning to the previous connection between RealTime wrappers and PCAN-USBs might be helpful. Another possible reason is due to the fact that 5 PCAN-USBs communicate with the workstation. There might be some conflicts between them and it is doubtful that so many devices can be handled in time. One better configuration could be the application of PCI instead of USB or replacing PEAK CAN adapters with Vector CAN adapters.

Another parameter 3σ is also calculated based on the timing output. In this work, the real time factor represents the factor of the time needed to calculate the current simulation step (including platform calculations) to the simulation time. For the RealTime wrapper, it is devised to act (exchanging data) according to the defined timeline. However, due to the feature of the non-real-time OS (MS Windows) and different kinds of jitter, real-time factor is unable to stabilize at “1”. The statistics of real-time factors for different wrappers during the co-simulation is exhibited in Figure 102. The initial 5 time steps of each wrapper are not counted in the statistics as the initial several time steps’ real-time factors are extremely high.

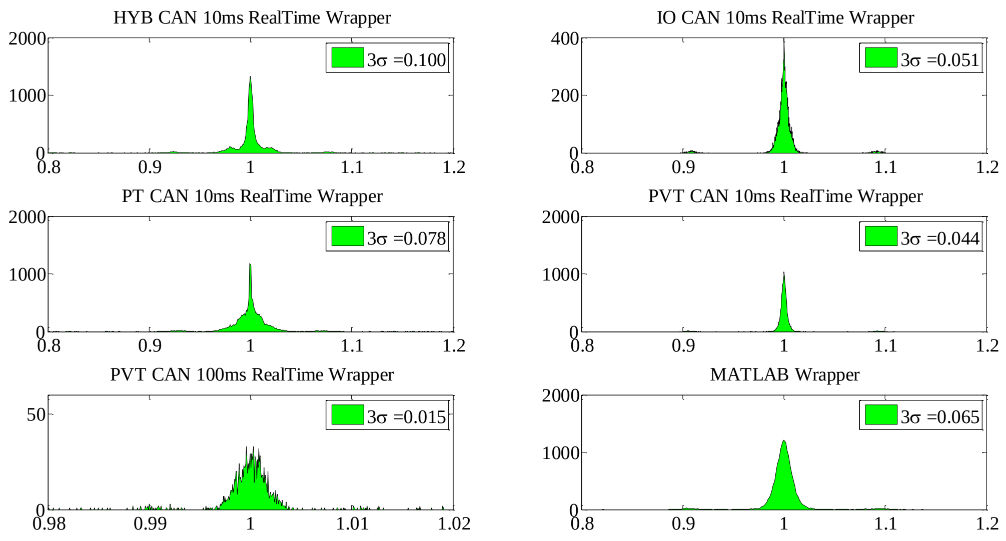


Figure 102: Statistics of the real-time factor of different wrappers

Figure 102 is created by histogram plot function in MATLAB. In each sub-figure, the histogram reveals the underlying shape of the distribution of the real-time factor of one wrapper during the entire co-simulation (except initial 5 steps). In the figure, the y-axis indicates the number of time steps while the x-axis denotes the real-time factor.

The distribution of real-time factor of each wrapper resembles a normal distribution. Thus, to simply the situation, it is assumed that the distribution of real-time factors subjects to normal distribution (Formula (6.5)).

$$X \sim N(\mu, \sigma^2) \tag{6.5}$$

where a random variable of real time factor X is distributed normally with mean value μ and variance σ^2.

The notation of the 3-sigma rule is utilized here to analyze the jitter. About 99.7% of values are within 3σ (three standard deviations). 3σ offers a statistical indicator to denote possibility of missing deadlines and the target is to minimize it. The smaller 3σ is, the less possibility of missing the deadline occurs. 3σ is also calculated and depicted in Figure 102.

However, it is also notable that once the co-simulation is implemented in a standard MS Windows with current hardware configuration, the real-time factor provided by MC is not totally trustable and missing the deadline is still possible - even if real-time factor is “1”, due to the reasons below (although some of them might be negligible):

- Jitter of the timer in MS Windows
- Execution time of processes
- Waiting time of the execution of processes
- Jitter in the communication with USB
- CAN bus delay
- Period of data transmission
- Period of data receiving
- Jitter of receiving

Figure 103 shows the total transmission time from calculation data ready for MC to data received by the HVCU. This total transmission time also denotes the dead-time in the closed control loop of HVCU and plant model. It is also notable that in a Windows OS the execution of processes is not immediate and probably has to wait.

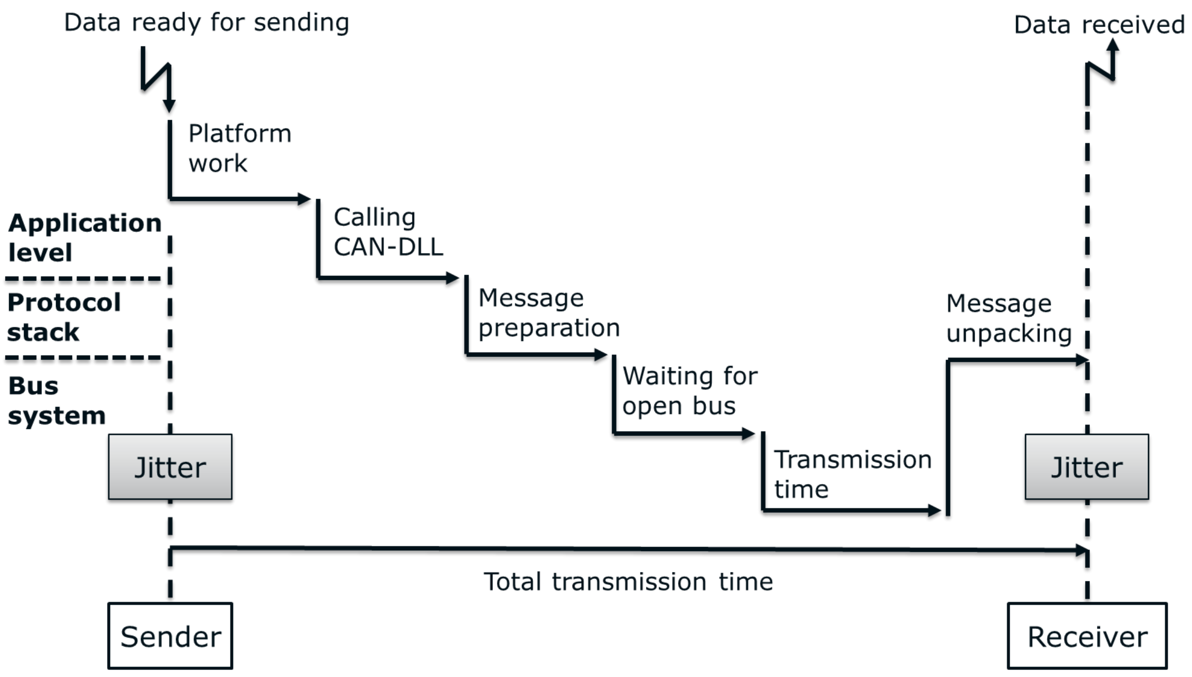


Figure 103: Total transmission time (dead-time) from the sender (MC) to the receiver (HVCU) [46]

In case that the calculation of the model is always faster than real-time， the combination of dead-time and jitter brings unpredictability to the test system, as shown in Figure 104.

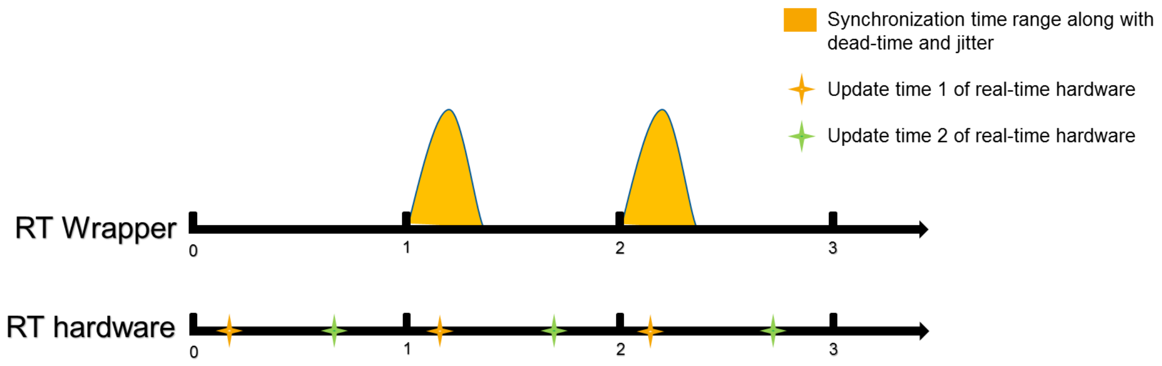

Synchronization time range along with dead-time and jitter
Update time 1 of real-time hardware
Update time 2 of real-time hardware
RT Wrapper
RT hardware
0
1
2
3

Figure 104: Unpredictability arising from time dead-time and jitter

As the dead-time and jitter in the developed test system are not negligible, the real-time hardware, which has update time “1”, is affected. Real-time hardware’s input signals sent from the plant model and its output signals received by the plant model are unpredictable in the time line. But the real-time hardware, which has update time “2”, is not affected by the dead-time and jitter. As a result, the repeatability of test results is not totally guaranteed.

After each test, there is an approach to quantify the dead-time and jitter by applying a watch dog (saw tooth) on several signals (like checksum signals). In future work, the watch dog shall be added, so the real time latency (dead-time plus jitter, and simulation slower than wall clock time in some cases) can be quantified and viewed as an indicator of the reliability of test results.

Assessment of real-time influential factors

Furthermore, another quantitative test is implemented in order to assess the real-time influential factors. One factor is implementation of measurement by CANape when the co-simulation takes place. Another factor is the setup of high priority of co-simulation related processes. Specifically, when the co-simulation starts to initialize, mysqld.exe, runKernel.exe, conhost.exe and icosm.exe are set to high priority in the Task Manager.

After implementation of the same test case, the average value of 3σ of 6 wrappers are calculated for 3 situations (Test with CANape measurement, normal test and test with high priority), as shown in Table 7.

Table 7: Comparison of average value of 3σ

Test type	Test with CANape measurement	Normal test	Test with high priority
Average value of 3σ	0.0238 (+22%)	0.0195	0.0192 (-1.5%)

From Table 7, it is known that other running programs can have an apparent impact on the jitter of co-simulation, especially when CANape is also a “real-time” software. It can display and monitor the CAN bus data or ECU internal signals in real time. So it is suggested that during the co-simulation, processes which are not related to MC need to be shut down. Also, setup of high priority lowers a little jitter (-1.5%) compared with normal test. In this case, the setup of high priority is not necessary but beneficial.

Investigations of a 605-second test case

To examine the long-period real-time behavior, the 150 seconds test case is extended to a 605 seconds test case and timing information is also obtained from timing output in MC.

The calculated 3σ of each wrapper in this test case is shown in Table 8.

Table 8: 3σ of each wrapper in a 605 seconds test case

Wrapper	HYB CAN 10 ms	IO CAN 10 ms	PT CAN 10 ms	PVT CAN 10 ms	PVT CAN 100 ms	MATLAB
3σ	0.855	0.846	0.859	0.839	0.741	0.876

The values of 3σ of each wrapper in 605 seconds test case is much higher in comparison with those in 150 seconds test case. It is conformed that there is no manual intervention during the co-simulation and this phenomenon is repeatable. The reason for the significant increase in 3σ lies in the fact that at some time steps the real-time factor is markedly higher. The dynamics of calculation real-time factor as well as real-time factor of all wrappers over time are shown in Figure 105. In each sub-figure, the x-axis denotes time, while y-axis denotes real-time factor or calculation real-time factor. After 190 seconds serious fluctuation of real-time factor occurs, which is as high as 5 and as low as 0.2. Therefore, the 3σ is considerably enlarged.

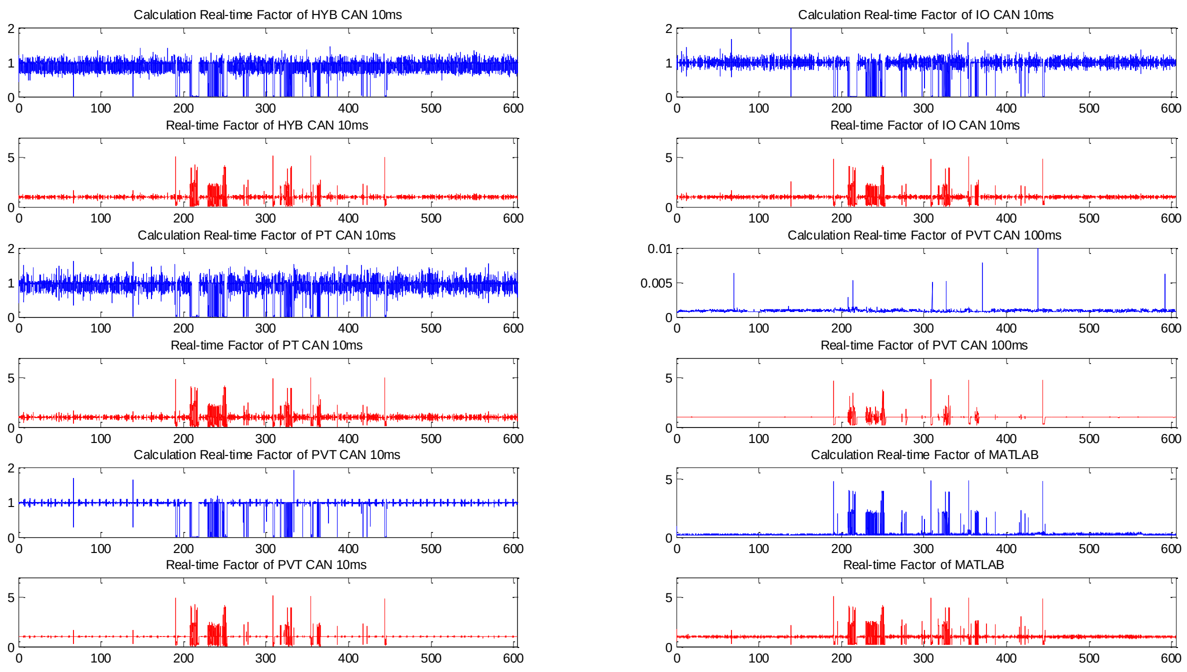


Figure 105: Calculation real-time factor as well as real-time factor of all wrappers

The reason for this is that after some time MATLAB occasionally calculates more slowly, at the same time PEAK programs calculate comparatively faster. It is possible that the co-simulation is handicapped by some other processes (like antivirus software) in Window OS or probably MATLAB is slowed down by itself. A more detailed insight into the mechanism of how MC program coordinates a co-simulation and recording of usage of CPUs is further needed.

The same test case has been tested for several times. The simulation time drift compared to wall clock time (calculation of MATLAB slower than wall clock time) randomly takes place with different severity and the jitter arising from Windows OS cannot be neglected. A permanent and clean solution might be a RTOS in which the co-simulation execution is possible to be hard real-time and problems resulting from the Windows OS can be remarkably mitigated. Also, running the C code complied from Simulink model has more real-time guarantee than Simulink model running in MATLAB. In addition, a new product Testbed.CONNECT developed by AVL List GmbH, which brings and connects simulation to the testbed and is hard real-time, can be further considered [47].

6.9.2 Reliability test

The reliability of the HVCU real test system signifies that the proposed test system can at least partially reflect the behavior of a real vehicle and can be used to detect a part of bugs in control software before the vehicle is prototyped. Thus, the comparison between the measured data from a real car and the test results from the HVCU real test, which has the same inputs, is not only significant but also necessary. In the Signal Builder, these four parameters need to be accordant with real driver inputs: PoET_DriverKey (driver key), PoET_PosnAccPedl (position of the accelerator pedal), PoET_PosnBrkPedl (position of the brake pedal), POET_PosnGearLvr (position of the gear level).

Fortunately, the HVCU has already been applied in a prototyped hybrid car. In one vehicle test, all CAN bus signals and part of essential internal signals of HVCU were measured and saved.

Based on the measurement, a new test case with driver inputs from the real vehicle test is created. In this test case, the position of accelerator pedal and brake pedal are generated in accordance with the real driver's action. These two signals in the test case range from 0, which indicates no pressing, to 100, which means full pressing. The position of accelerator pedal is captured by two sensors, which sense it out to the HVCU as two analog inputs, and finally HVCU transmit these signals on CAN buses. One point that should be paid attention to is that accelerator pedal position can be indicated by two CAN signals on two different CAN Buses (PT CAN and PVT CAN) and their physical values are different due to the calculation method. The CAN signal on PT CAN bus can be directly utilized as the real position of accelerator pedal.

Another difficulty is that in the real vehicle only the pressure of the brake pedal (unit: bar) and the state of the brake pedal (pressed or un-pressed) are recorded. In the real vehicle test, the pressure of the brake pedal varies from 0 bar to 80 bar. In CRUISE simulation of the plant model, the brake pedal pressure raw value is calculated based on the input of brake pedal position, but brake pedal pressure raw is not sent to the HVCU via CAN bus. The relationship between brake pedal positon and brake pedal pressure raw is found and according to the brake pedal pressure raw from the real vehicle, the corresponding brake pedal position in inputs is obtained.

Meanwhile, the position of the gear level is also recorded via a CAN bus. In the Signal Builder, if POET_PosnGearLvr is 1, 2, 3 or 4, it means Parking, Reverse, Neutral and Drive respectively. Another important change for the test case is that in the real vehicle the Clamp 50, which brings the powertrain system into operation, is activated before the measurement. In the test case, during the time period when the brake pedal is firstly pressed, the Clamp 50 shall be "1" and therefore the powertrain system is correctly started up. In the signal builder this exactly denotes that POET_DriverKey equals to "2". For other necessary inputs for the plant model, the values from the NEDC test case are adopted.

Figure 106 shows the generated inputs based on the real vehicle measurement.

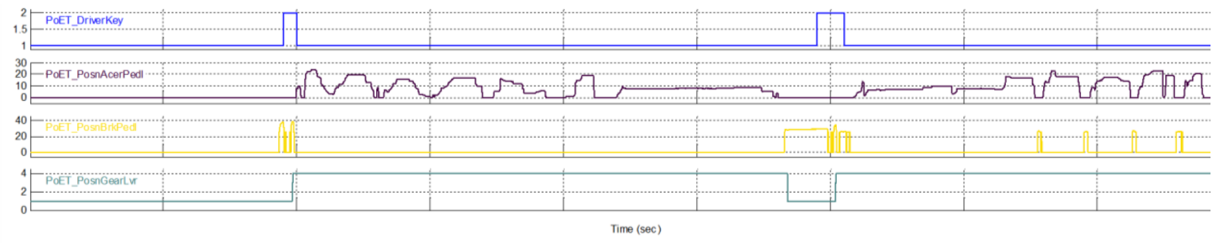


Figure 106: Generated inputs from the real vehicle measurement

Although the inputs to the plant model are correct, it does not necessary mean that the HVCU can obtain the correct signals. This is because the inputs need to pass through the plant model, ICOS interfaces, CAN bus and finally arrive at the ECU. And for accelerator pedal sensor signals generated by the accelerator pedal positon, they need additional digital-analog conversion.

The test case was tested on the developed HVCU real test system and CAN signals of PT CAN, HYB CAN and PVT CAN as well as some important internals signals are measured and recorded by CANape.

Figure 107 shows the comparison between input signals from the HVCU in the real vehicle test and those from the HVCU in the MC test.

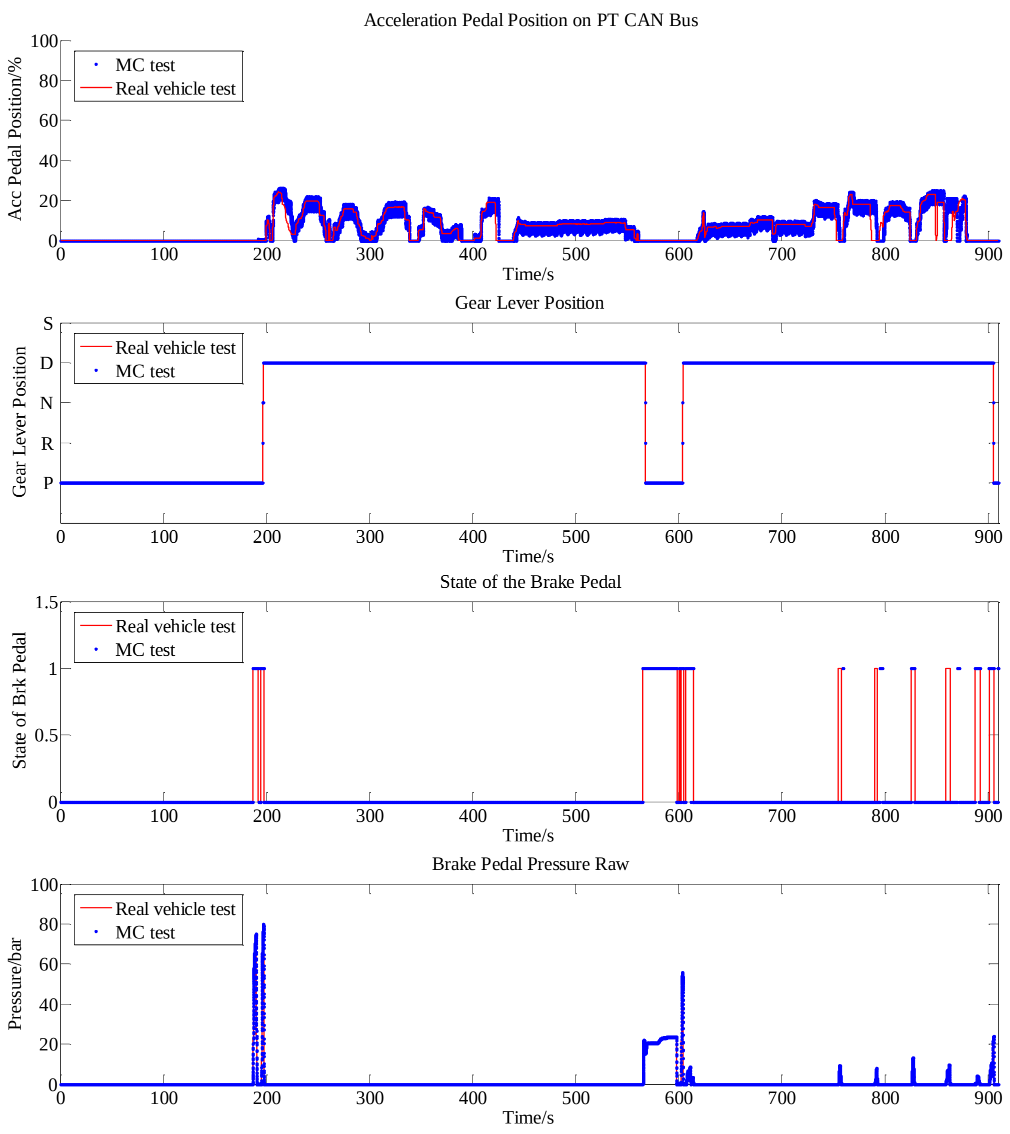


Figure 107: Comparison between input signals of the HVCU in the real vehicle test and those of the HVCU in the MC test

In general, the simulated signals obtained by the HVCU reflect the real signals in the real vehicle but some of them are not fully satisfying. In light of the accelerator pedal, the signal quality is degraded by noises. This is due to the fact that two real solenoids controlled by the HVCU are applied in the current arrangement. Meanwhile, the HVCU with solenoids and PEAK modules (MicroMods) utilize the same power supply. Once solenoids start to work, the output of power supply becomes unstable and thereby outputs of MicroMod are degraded.

The idea of separated power supply of the HVCU and MicroMods has been tested but proves to be unfeasible. This is because the HVCU and MicroMods have to use the same ground. A possible solution is removing the solenoids and making solenoid simulators or finding a more stable power supply. In addition, it can be found in the picture that especially at the end of the test, signals from the MC test are markedly later than those from the vehicle test. This is because the slow-down of calculation happens in this 905 seconds test case and brings obvious time latency (about several seconds).

Considering the gear level position and the brake pedal state, they generally follow the line of real vehicle test. But at the final stage of the brake pedal state, some mismatch of signals can be found, which also results from the time latency.

Different from previous signals, in the MC test the brake pedal pressure raw value is only calculated in the plant model but not sent to the HVCU. So the signal in the plant model is compared with the real signal of the vehicle test and they can be exactly matched.

After the input comparison, it is known that input signals in the MC test can generally reflect the real inputs in a car. Thereby some other important signals, which indicate the general performance, are further compared.

The hybrid vehicle that is being investigated comprises totally 7 sorts of hybrid operation modes. Figure 108 shows the differences between the hybrid mode in the real vehicle test and MC test. The description of each hybrid mode is shown in Table 9.

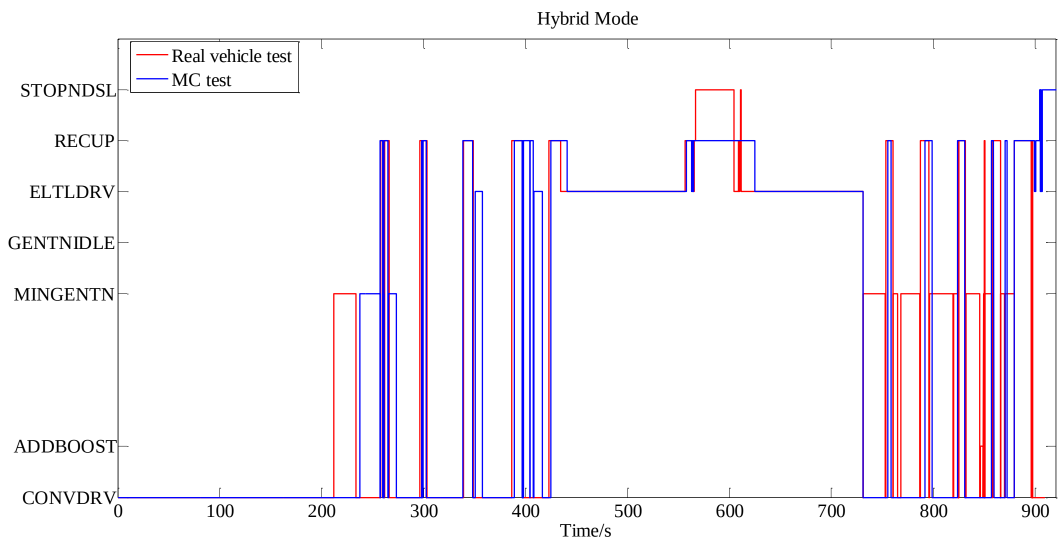


Figure 108: Comparison of hybrid mode between the real vehicle test and MC test

Table 9: Abbreviation and corresponding meaning of hybrid modes

Abbreviation	*Meaning*
STOPNDSL	Engine stop at vehicle standstill
RECUP	Recuperation (regeneration of brake energy)
ELTLDRV	Pure electric driving
GENTNIDLE	Generation at idle (charging the HV battery at vehicle standstill; engine is running at idle speed)
MINGENTN	Minimum generation (charging the HV battery during driving by load point shifting)
ADDBOOST	Additive boost (using EM to apply positive torque to assist the engine for propulsion)
CONVDRV	Conventional driving (engine only)

From the beginning, the hybrid mode of the MC test is totally different from it of the real vehicle test. One major reason for this is that in the MC test, due to the limitation of the plant model, only the conventional start is implemented while the real vehicle might adopt impulse start. Another reason is due to the dissimilar SOC values. In the plant model all battery cells' voltages are idealized as constant, which influences the calculation of SOC values. Also, the initial SOC in the MC test differs from the real initial SOC in the real vehicle test. This might influence the decision-making of the HVCU about selecting the optimal hybrid mode. In addition, between 250 seconds and 400 seconds, the MC test can partly reflect the hybrid mode variation between CONVDRV and MINGENTN of the real vehicle.

Figure 109 shows the actual gear comparison between the real vehicle test and MC test. At the beginning of the test, the actual gear change of MC can follow it of the real vehicle. But with time passing, the behavior gradually becomes apparently different. Gear selection is implemented by the TCU in a real car. One important reason for the variation is the difference between the real TCU and the virtual TCU in the plant model. The calibration of the virtual TCU differs from it of the real TCU in a car. In addition, gear selection is based on a shift map, which has two input signals, vehicle speed and driver torque demand on the wheel (or accelerator pedal position). And the selection of the shift map is according to the current hybrid mode. As deviations of vehicle speed, accelerator pedal position and hybrid mode of the MC test from the real vehicle test are accumulated over time, the behavior of actual gear in the MC test is gradually different from it in the real vehicle test.

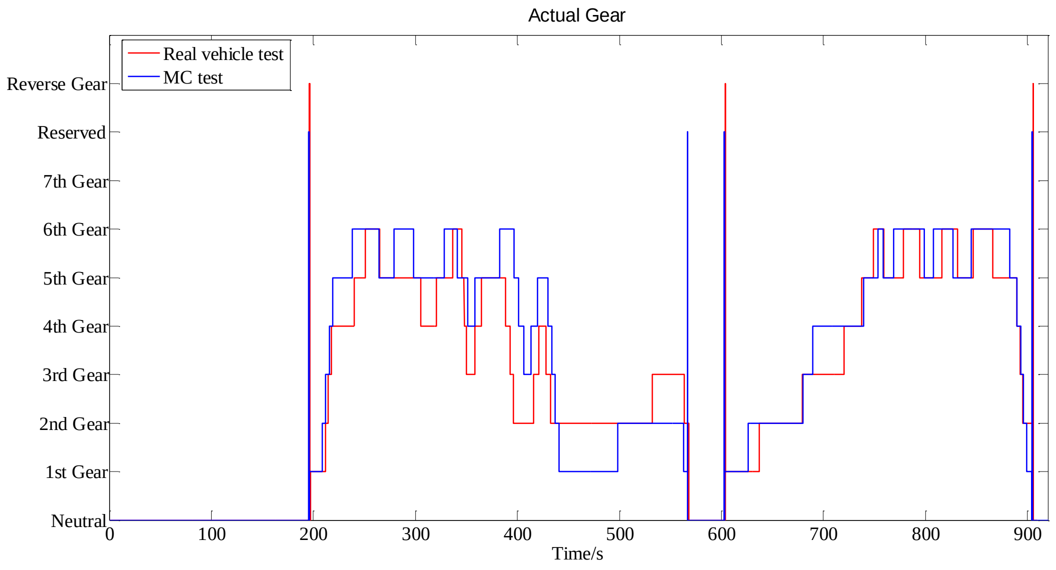


Figure 109: Comparison of actual gear between the real vehicle test and MC test.

Figure 110 shows the vehicle speed comparison between real vehicle test and MC test.

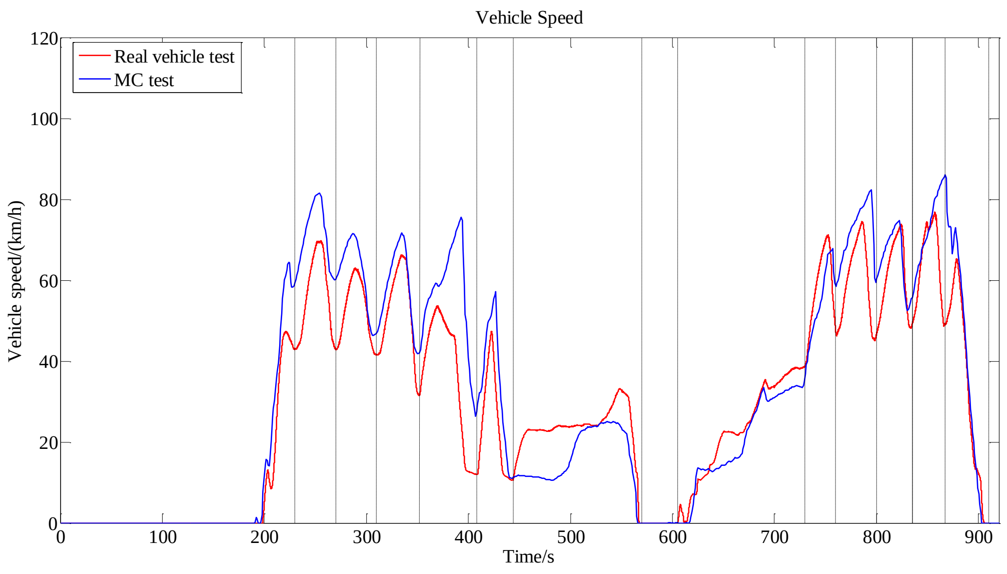


Figure 110: Comparison of vehicle speed between the real vehicle test and MC test

The vehicle speed trend of the real vehicle test and it of the MC test are generally similar, but between 450 seconds and 550 seconds they are obviously different. Besides the influencing factors discussed before (noises in inputs, time latency, different powertrain start approach and SOC), the plant model is also required to be improved and requires a more mathematically precise description of the real vehicle.

Another reason for the deviation might be that the communication topology between ECUs, sensors and actuators in MC test differs from it in a real vehicle. Two examples are given in Figure 111 to show the variations.

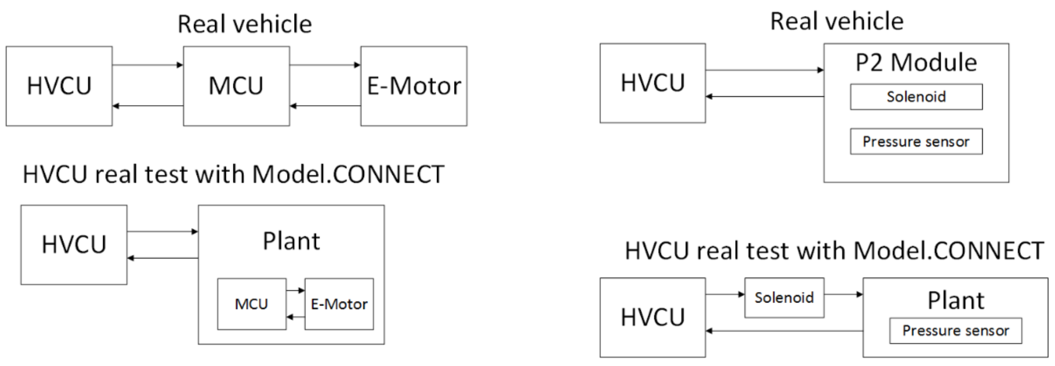


Figure 111: Differences between communication topology of the MC test and a real vehicle

6.9.3 Repeatability of test results

Repeatability of test results denotes that if one test case is implemented for many times, some important signals in each test should have nearly same trend and behavior over time. To evaluate the repeatability of test results in the proposed test system, a test case, which lasts for 400 seconds has been created. In this test case the driver drives up to 60 km/h and frequently accelerates and decelerates to a crawling speed, which poses a challenge to the control algorithm. This test case has been tested and recorded for 10 times. In every repeatable test, the driver inputs (accelerator pedal position and brake pedal position) are same. Vehicle speed is one of the most important control objects of the vehicle and is utilized to assess the repeatability of test results. The vehicle speed curves of 10 tests are pictured in Figure 112.

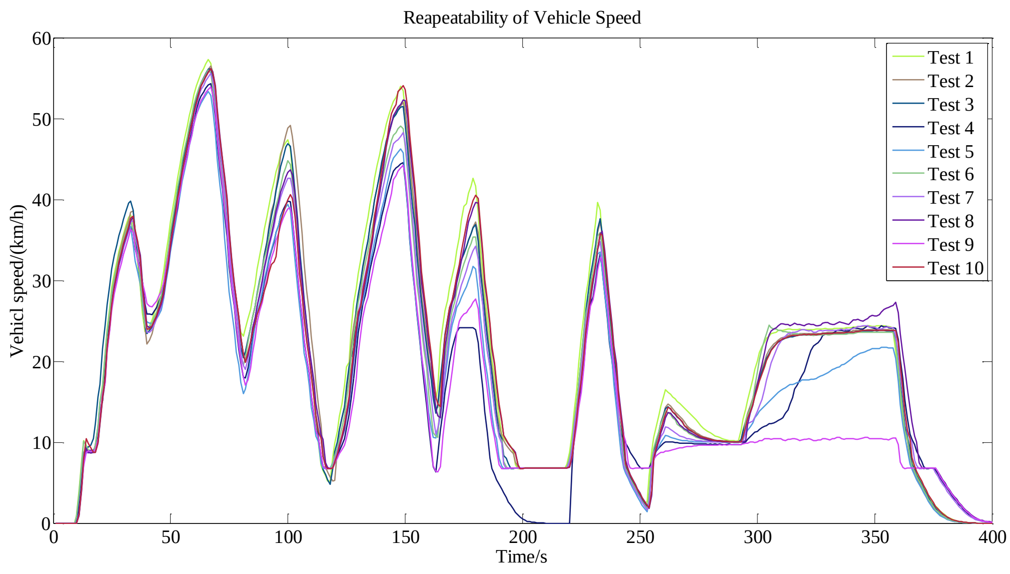


Figure 112: Vehicle speed curves of 10 tests

According to Figure 112, initially all curves are almost overlapped but the variations among 10 tests grow along with the simulation time. At the second half of the time range, several tests behave quite differently from the most. The most possible reason for this is the combined effect of dead-time, jitter and simulation time drift compared to wall clock time. As it was tested in automation, the log files and timing output were overwritten. It is unknown whether in some test results the serious simulation time drift takes place or simply jitter and dead-time exists. Also, the degradation of outputs from MicroMods brings much uncertainty.

For further improvement of the repeatability, the most attention should be attached to the augmentation of the real-time condition of the proposed test system as well as the signal quality of I/O simulation.

6.9.4 Applicability for various test cases

In order to assess the applicability of the test system as well as the plant model, the NEDC test and random tests are introduced and investigated. Moreover, a special test, which is related to power-up-and-down function of the HVCU, is also implemented.

NEDC Test

The New European Driving Cycle (NEDC), which was last updated in 1997, is a driving cycle delegated to evaluate the emissions and fuel consumption in passenger cars [48].

In the test case of NEDC, a desired curve of the vehicle speed is sent to the CRUISE module inside the plant model. For the purpose of following the desired vehicle speed, CRUISE calculates the position of brake pedal and accelerator pedal based on a look-up table, which simulates the behavior of a driver.

The comparison between the desired vehicle speed and vehicle speed in the HVCU real test with MC can be found in Figure 113.

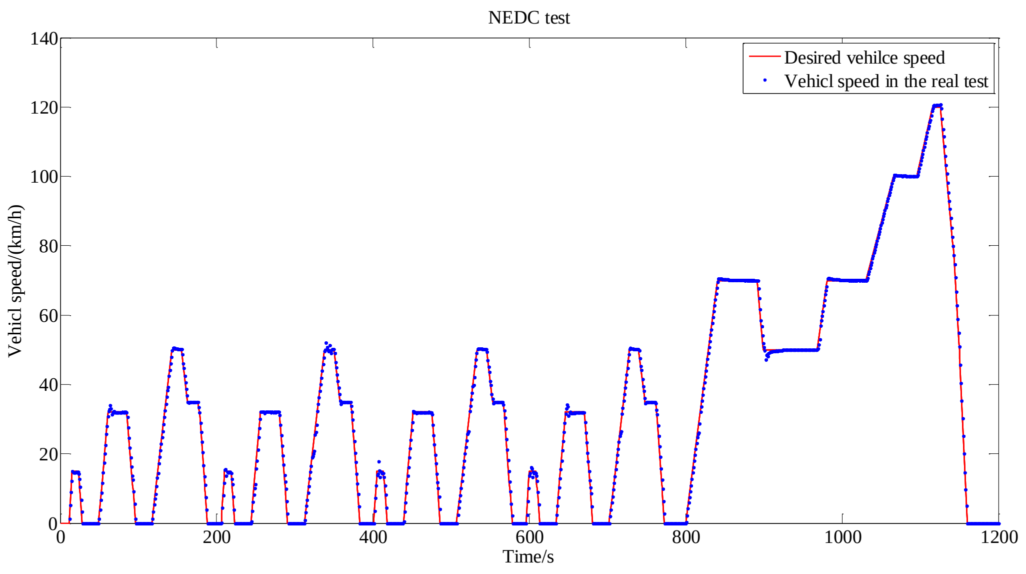


Figure 113: NEDC test results

Except some minor overshoot when vehicle speed changes aggressively, in most time the actual vehicle speed could exactly overlap the desired vehicle speed. NEDC test is able to cover most operation modes of the hybrid vehicle, which proves that the developed test system as well as plant model is applicable under complex test procedures.

Random test

By the usage of Monte Carlo method, random test cases for three different traffic scenarios are created. The traffic scenarios include: city scenario with heavy traffic, in which the vehicle speed ranges from 0 to 30 km/h and the car usually drives idly, city scenario with good traffic, in which the vehicle speed ranges from 0 to 60 km/h, and highway scenario, in which the vehicle speed ranges from 80 km/h to 120 km/h.

The inputs setup is the same as in the NEDC test, except the desired speed.

The vehicle speed comparison between the desired vehicle speed and the vehicle speed in the HVCU real test for city scenario with heavy traffic, city scenario with good traffic and highway scenario are shown in Figure 114, Figure 115, and Figure 116 respectively.

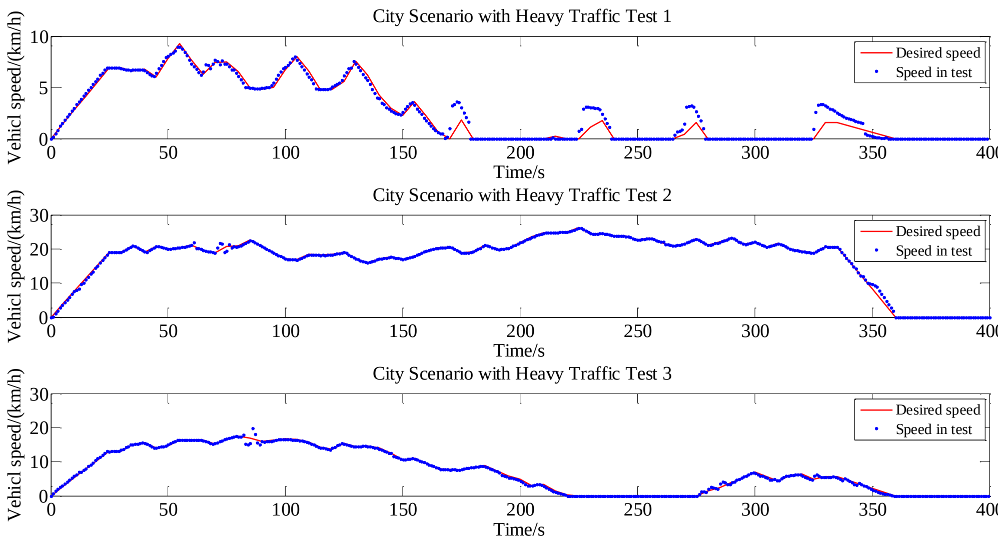


Figure 114: City scenario with heavy traffic

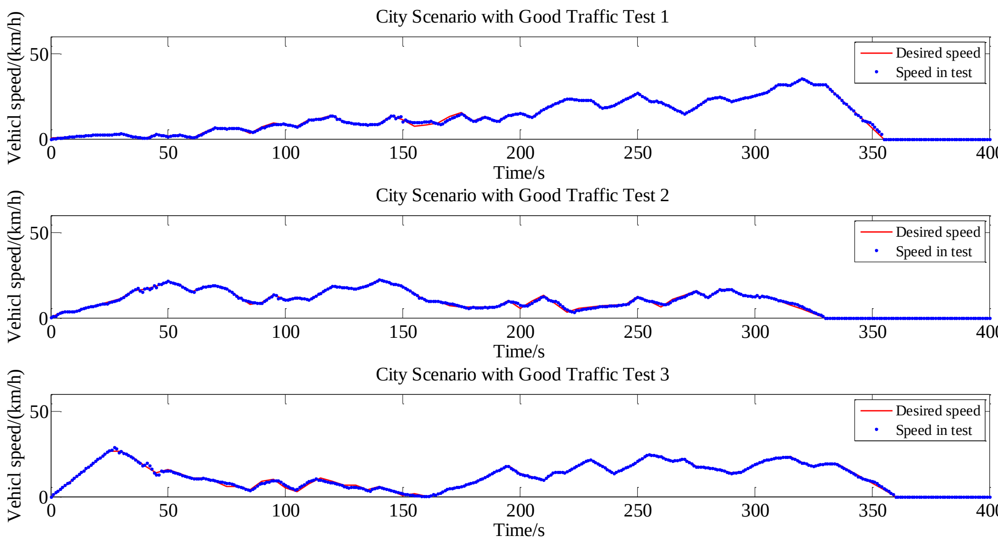


Figure 115: City scenario with good traffic

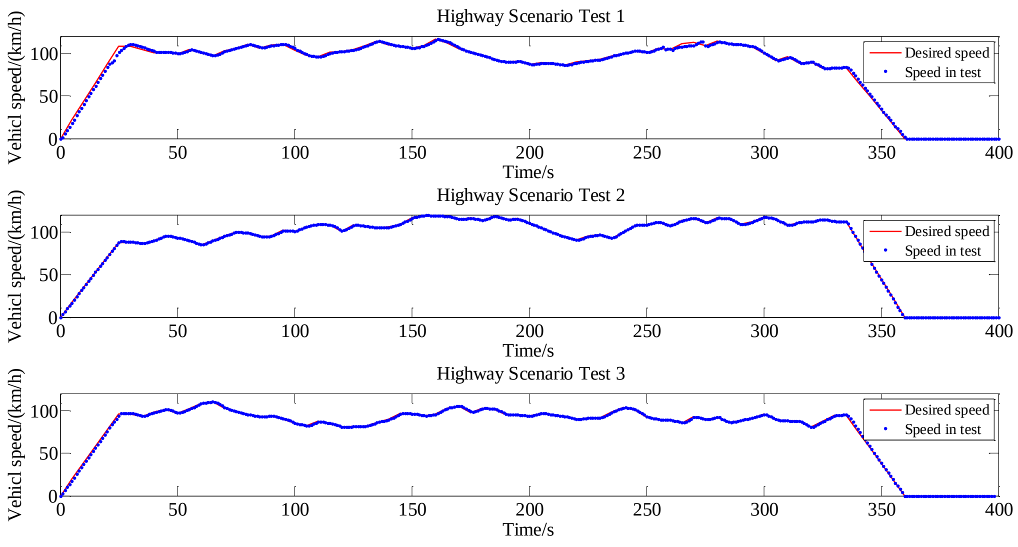


Figure 116: Highway scenario

These figures show that the vehicle speed in those tests can correctly follow the desired speed curve with only a few variants in different test scenarios. In the first test in Figure 114, when the vehicle stands still and then crawls (less than 2 km/h), there is an obvious overshoot of the test result. But this is understandable because it is exceedingly difficult for the control unit to control the vehicle in plant model under such low speed. This is also related to the calibration of the driver model inside CRUISE.

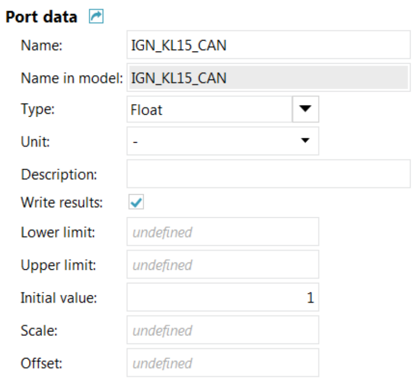
In addition, it can be found that at the left side of the second and third test in Figure 114, a small number of signals are suddenly obviously different. The most possible reason is that the time latency around that time happens.

To summarize, the random test shows that the test system as well as the plant model is applicable in various scenarios. Also, even though during the test the dead-time and jitter is unavoidable, the control algorithm is robust to some extent and it controls the vehicle speed generally appropriately.

Power-up-and-down test

Power-up-and-down function is one of the most critical functions of the HVCU. It is triggered by certain conditions and should be implemented according to a defined procedure. The variation of conducting such test is that during the MC co-simulation the control unit has to be shut down, after which no CAN message from the control unit is sent out. Moreover, after the initialization phase of MC, the control unit shall be power-on (sending out CAN signals) due to the working process of the MC project containing RealTime Wrappers. When the co-simulation in MC is started, models and components of MC are initialized at the beginning, after which MC sends out CAN messages with initial values and waits for feedback. Until when MC receives CAN messages as defined in *.ini file, the co-simulation can really move on.

Due to the reasons mentioned above, the test procedure of such kind of test shall be a little different from other test procedures. Switch-on and switch-off of the HVCU is controlled by a signal called "Clamp 15". In the MC project, this initial value of this signal is defined as "1", which means "switch-on", as shown in Figure 117. Thus, the HVCU can be switched on after the initialization of MC and feed CAN messages back to MC.



Port data

Name:	IGN_KL15_CAN
Name in model:	IGN_KL15_CAN
Type:	Float
Unit:	-
Description:	
Write results:	✓
Lower limit:	*undefined*
Upper limit:	*undefined*
Initial value:	1
Scale:	*undefined*
Offset:	*undefined*

Figure 117: Setup of initial value

Another necessary change is implemented in the test case inputs. In the original test case, the PoET_DriverKey was “0” at the beginning of the test, which means Clamp 15 is “0”. So, at the beginning the shut-down of the control unit is accomplished not at the dcided time point. Therefore, the original test case is changed as Figure 118 shows.

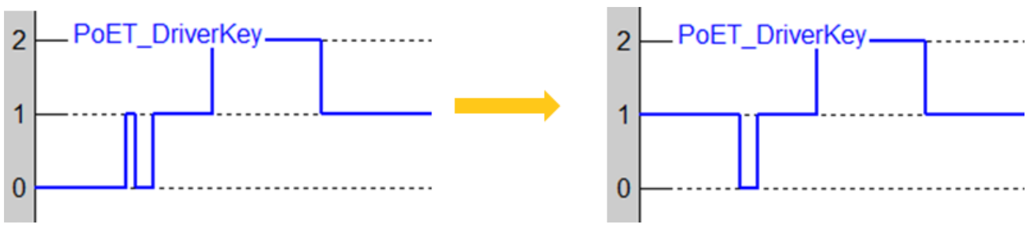


Figure 118: Change of PoET_DriverKey (Clamp 15)

Test case 14-2 is tested in the developed HVCU real test environment. This test case is about starting ShutDown during WakeUp state because of not pressing the brake pedal. This test case aims to check the hybrid system state transition from INIT to CU_SHTDN due to the status of the brake and status of Clamp 15 signal. The test result of the hybrid system state transition is shown in Figure 119 and the requirements are fullfilled. The description of each hybrid system state is shown in Table 10.

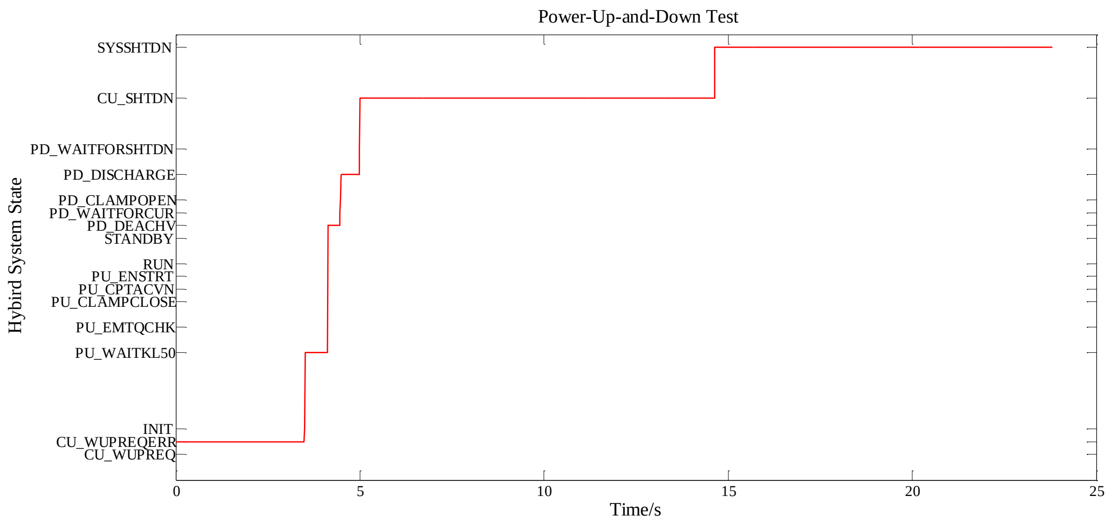


Figure 119: Test results of test case 14-2

Table 10: Abbreviation and corresponding meaning of hybrid system states

Abbreviation	*Meaning*
CU_WUREQ	Control unit wakeup request
CU_WUPREQERR	Control unit wakeup request error
INIT	Initialized
PU_WAITKL50	Power up wait for clamp 50
PU_EMTQCHK	Power up EM torque check.
PU_CLAMPCLOSE	Power up HV clamp close
PU_CPTACVN	Power up component activation
PU_ENSTRT	Power up engine start
RUN	Hybrid system is Running
STANDBY	Standby state when clamp 15 is deactivated during driving
PD_DEACHV	Power down deactivation
PD_WAITFORCUR	Power down wait for current
PD_CLAMPOPEN	Power down HV clamp opening
PD_DISCHARGE	Power down discharge
PD_WAITFORSHTDN	Power down wait for shutdown
CU_SHTDN	Control unit shutdown
SYSSHTDN	HVCU shutdown state

6.9.5 Summary of the applicability of the HVCU real test platform

The applicability of the proposed test system is summarized with respect to different real-time requirements.

The test tasks, which require the soft real-time condition, can be implemented in the proposed test system without any problem, as the occasional time latency brings no damage to the test results. These test tasks include:

- Interface check (e.g. Electrics\Electronics check and Component interface check)
- Sequence check (e.g. Power-up-and-down test)
- Stationary function check

The test tasks, which require firm real-time condition, can also be implemented in the proposed test system, which is validated by practical usage. These tasks contain:

- Short-period function check
- Plant model check
- Test cases with a driver model (NEDC test and Random test)

For a short-period test case, the rare occurrence of missing the deadline might not cause a noticeable consequence. Also, during the practical utilization, the plant model has been debugged and improved. In light of the test case with a driver model, during the test plant model can be controlled following the desired vehicle speed. This shows that the control algorithm is robust and can accept a limited degree of unpredictable time latency. So the control unit's overall performance can be evaluated. However, the detailed behavior of the control unit is possibly different each time with a same test case.

For those test cases, which require hard real-time condition, the proposed test system currently is not reliable. These tasks involve:

- Long-period function check
- Generic test

This is because in the test system there always exists risk of unpredictable time delay and the repeatability of test results is not at least 95%. The problem becomes more critical with longer simulation time.

In practice, when the user detects a bug during the general HiL testing, he or she has to find out the problem source, due to the control system itself as well as plant model or due to the time latency. Additionally, to ensure that the bug is repeatable, the user has to run the test case for several times.

6.9.6 Ease of testing

The efforts of implementation of a test case in HVCU real test platform are satisfying (Table 11).

Table 11: Time of each work during testing

Work	*Time*
Loading and modification of a new test case in Simulink	3 minutes
Generation of a new job submission in MC (without running)	3 minutes (to be improved)
Initialization for run or rerun of the job submission in MC	3 minutes

Furthermore, the JMS of MC is a convenient tool that is able to schedule tests and preserve test results. It also enables running of a batch of test cases without manual intervention, in combination with the scripts in CANape, as described in chapter 6.7.2.

In addition, after one test, the improvement to the plant model in MATLAB can be immediately implemented. If ICOS interfaces or the model name are not changed, the MC project can be directly reused.

6.9.7 Ease of setup for further projects or different ECUs

One outstanding challenge for setup of the HVCU real test platform is the simulation of I/O and its related tuning work. It makes little sense that for a new ECU the hardware setup has to be configured from the ground up again and some supplementary circuits have to be soldered.

One solution is providing a configurable load board comprising generic sensor and actuator simulator as well as driver circuit, which is applicable to most ECU I/Os. This is possible, as I/O ports of ECUs have following common features:

- Limits of types (analog, digital, frequency or PWM)
- Limits of ranges (normally like 5 volts, 12 volts)

Therefore, a universally applicable configurable load board for simulation of I/O of different kinds of ECUs can be realized, which is able to reduce a large amount of effort concerning I/O simulation.

Another trouble is the generation of ICOS interfaces in Simulink models and connection of lines in MC project. Since some MATLAB scripts have been written, based on which simple GUI programs can be created and implement such tasks. Therefore, users can handle these tasks without deeper previous experiences.

Moreover, with additional programming, it is possible to directly generate AD/DA configuration file directly from ICOS interfaces, which could save an amount of time and can be as convenient as the commercialized products on the market.

The potential of ease of setup with assistance tools is described in Figure 120.

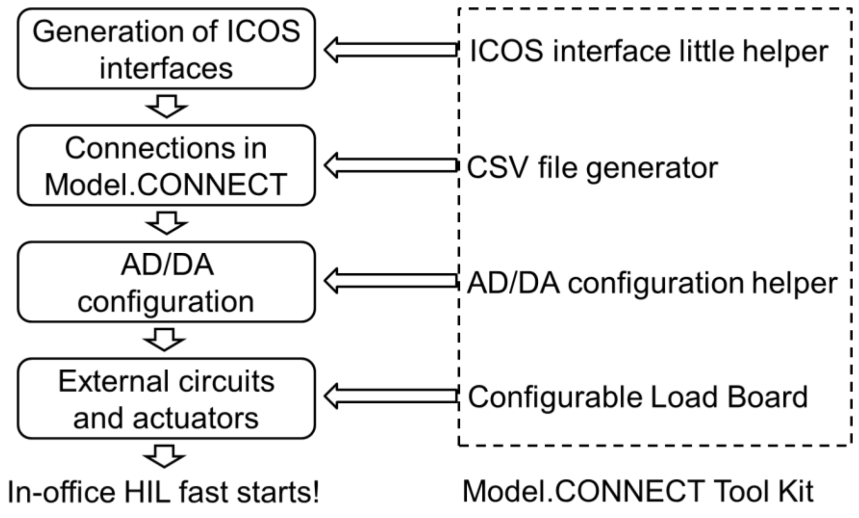

Generation of ICOS interfaces
Connections in Model.CONNECT
AD/DA configuration
External circuits and actuators
In-office HIL fast starts!
ICOS interface little helper
CSV file generator
AD/DA configuration helper
Configurable Load Board
Model.CONNECT Tool Kit

Figure 120: Potential of ease of setup with assistance tools

6.9.8 Possibilities for extension

The developed test system has high flexibility for extension. Once the real-time problem is solved, more virtual or real control units, actuators as well as sensors or offline systems can be supplemented to the current test platform. One prospective application with a real TCU is shown in Figure 121.

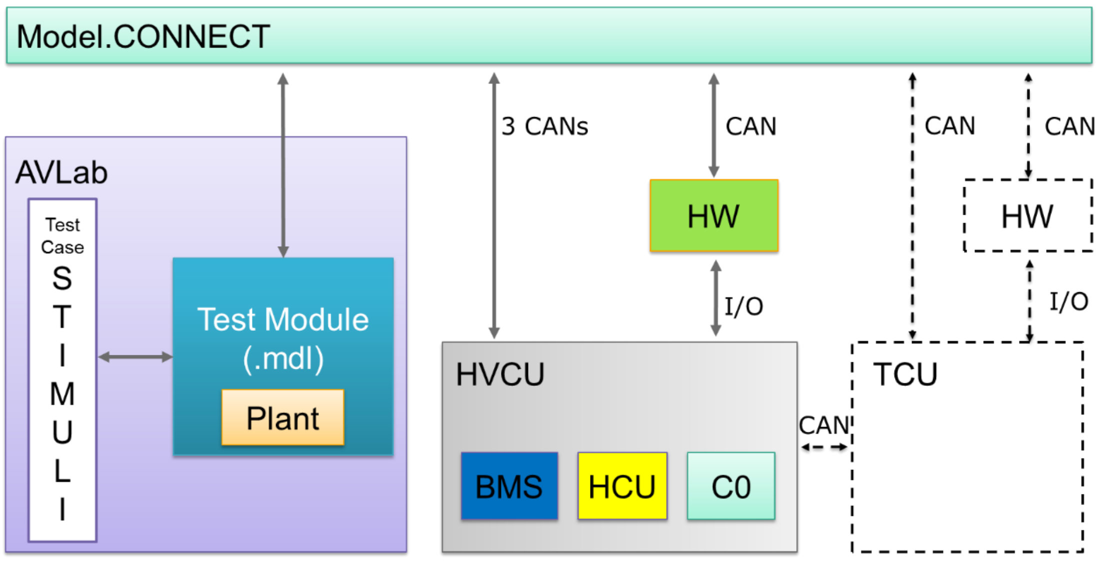


Figure 121: One prospect of application with a real TCU

In addition, it is realizable to integrate ADAS simulation tools (like PreScan) into the proposed test system (Figure 122). This is because MC provides interfaces to different tools and domains and the possibility of interfaces is increasing. Testbed.CONNECT and high-quality I/O modules are recommended in order to guarantee the real-time condition and good I/O simulation.

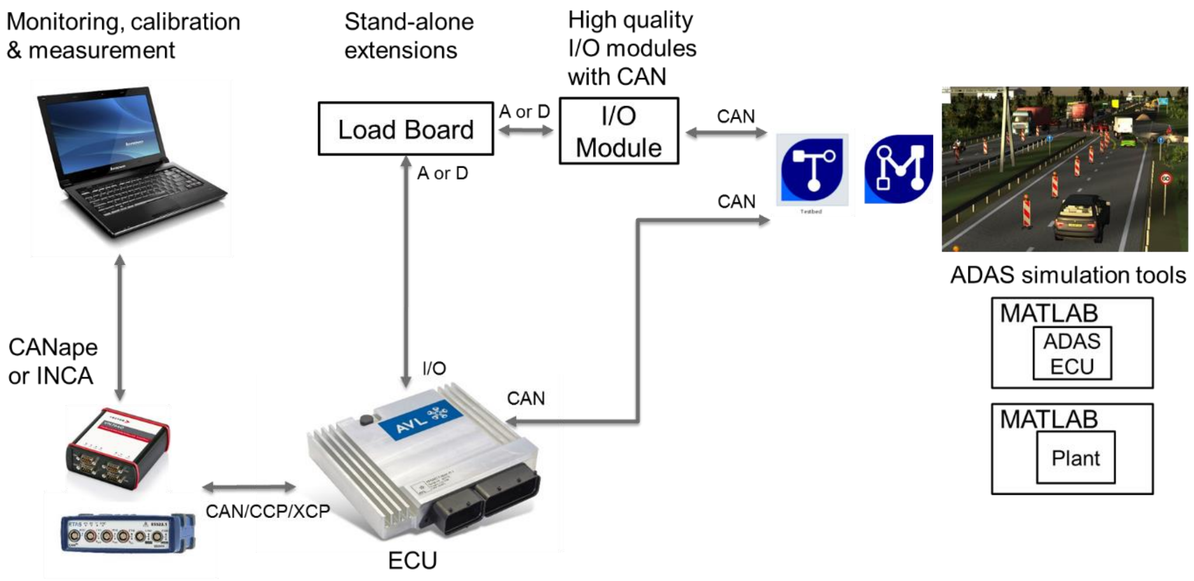


Figure 122: Outlook of integrating ADAS simulation tools into the proposed test system

6.9.9 Main challenges and solutions

The existing main challenges and corresponding solutions are summarized in Figure 123.

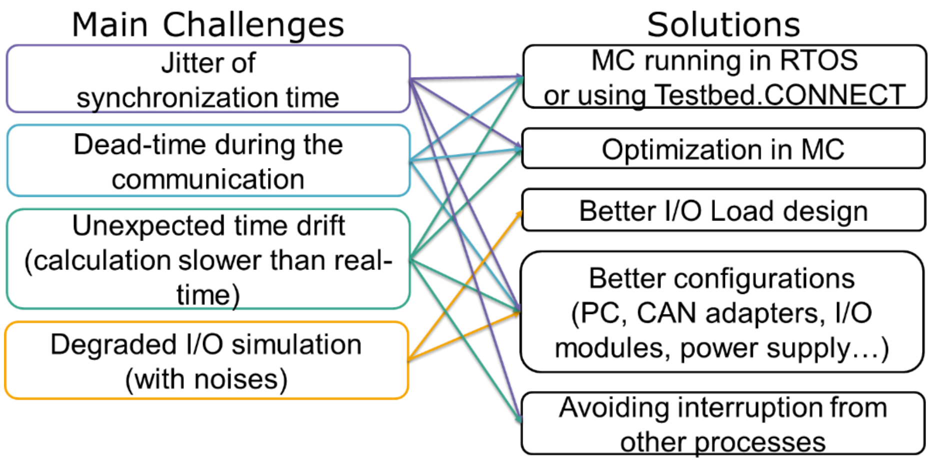

Main Challenges
Jitter of synchronization time
Dead-time during the communication
Unexpected time drift (calculation slower than real-time)
Degraded I/O simulation (with noises)
Solutions
MC running in RTOS or using Testbed.CONNECT
Optimization in MC
Better I/O Load design
Better configurations (PC, CAN adapters, I/O modules, power supply…)
Avoiding interruption from other processes

Figure 123: Main challenges and corresponding solutions

The optimal solution is the utilization of a RTOS. It could be a part of real-time processes of MC running in RTOS or using Testbed.CONNECT, which runs in a RTOS and comprises interfaces with MC and CAN or I/O communication. In this way, most real-time problems can be solved.

Another solution is optimization in MC, which is beneficial to mitigation of the influence of missing the deadline. In this work, the extrapolation method for all signals is zero-order-hold (ZOH). The first-order-hold (FOH) might be better in terms of linear varying signals, like vehicle speed. And MC provides compensation approach for time latency. For example, NEPCE (nearly energy-preserving coupling element) can significantly increase the accuracy of co-simulation results in one use case [21]. However, it takes huge efforts to apply these methods to this project as there are probably 50 important signals that need to be considered.

Better I/O load design refers to the utilization of a simulation circuit instead of two real solenoids, which make the power supply output unstable and degrade the output signals of MicroMods. Additionally, MicroMod has a problem with signal linearity and stability, so it is also important to check the quality of other outputs of MicroMod as well as inputs to them. Probably additional circuits will be needed to reduce noises.

Better configurations can help to relieve all existing problems, which includes a workstation with better configurations, better CAN adapters like Vector products or PCI CAN adapters, better I/O modules, and utilization of a power supply that is more stable.

Interruption from other processes in Windows OS has an apparent impact on the simulation time drift compared to wall clock time as well as jitter of synchronization time. Deactivating the antivirus software and other unrelated software is significant and it is recommended that during the co-simulation the workstation should be undisturbed.

6.9.10 Conclusion

In this chapter, seven criteria for assessment of the developed test platform are proposed. In light of the real-time performance, two indicators (the proportion of real-time factor of calculation more than "1" and 3σ of the real-time factor) are proposed. By looking into these two parameters and the dynamic variation of real-time factor as well as calculation real-time factor, the hard-real-time condition of the test platform cannot be guaranteed especially in a long-period (more than 150 seconds) test case. With the same driver inputs, the test results from the MC test are compared with those from the real vehicle test. It is found out that the proposed test platform can partly reflect the real situation of the vehicle and it has reliability to some extent. However, the repeatability of test results in the proposed test system is not satisfying but it has the potential to be improved. In addition, the overall test system with the plant model shows applicability for various test cases as they work well under different operation conditions and accept a special kind of test case, power-up-and-down test.

The proposed test system has a potential to be applied in industry, as setup procedure can be simplified and the approach of testing is convenient and partly automated. In addition, it has good scalability and more components can be integrated into the proposed test platform.

7 Work summary and outlook

In this thesis, testing and investigations of a hybrid control system in virtual and real test platforms using Model.CONNECT is implemented. The test platforms include:

1. TCU virtual test platform

2. Multi-ECU virtual test platform

3. HVCU real test platform

The TCU virtual test platform denotes the effect of Model.CONNECT co-simulation on the test results in comparison with the previous test environment. Co-simulation with MC always brings extrapolated errors, no matter if the co-simulation is parallel or sequential. In general, the overall extrapolated errors in a parallel co-simulation are more than an appropriate sequential co-simulation (which means that the sequence is similar to the original simulation), but less than the inappropriate sequential co-simulation. In theory, the simulation time of a parallel co-simulation is less than a sequential co-simulation, and the simulation time of Model.CONNECT test is less than it of the original test. However, in this work the sequential or parallel co-simulation has little effect on the overall simulation time as one model takes up most calculation time, and Model.CONNECT co-simulation takes more time than the original simulation as it needs time for initialization.

In the multi-ECU virtual test, new interfaces between ECUs and a newly merged plant model are created, and the multi-ECU virtual test platform is set up. Also, integrating Model.CONNECT into the existing tool chain is realized. However, the simulated vehicle in the co-simulation cannot really run as the stalling of engine takes place during the start-up stage. The most possible reason for the engine stalling behavior is caused by wrong calibration of the control units, but there also exist other possibilities. In the future, the control path of the start-up function needs to be further investigated.

In the HVCU real test, the basic test platform is developed, optimized and extended. Seven criteria are proposed to evaluate the developed test system, as shown in Table 12.

Table 12: Evaluation of the HVCU real test system

Criteria	*Evaluation*
Real-time condition	Soft real-time
Reliability	Partly reflecting the behavior of real car
Repeatability of test results	Bad
Applicability	Applicable for various tests
Ease of testing	Good and partly automated
Ease of setup	Potential to be easy
Possibilities for extension	Good

In future work, attention should be attached to the enhancement of the real-time capacity of the proposed test system. The best way might be applying an RTOS, but optimization in MC, better configurations as well as avoiding interruption from other processes can also relieve problems during a test, including jitter, dead-time and simulation time drift compared to wall clock time. Also the I/O simulation needs to be improved in the future.

List of Figures

List of Tables

Bibliography

[1] S. Davis, S. Diegel and R. Boundy, “Transportation energy data book”, Oak Ridge National Laboratory, Edition: 30, 2011. [Online] https://info.ornl.gov/sites/publications/files/Pub31202.pdf. [Accessed: 17- Dec- 2017].

[2] S. Williamson, “Energy Management Strategies for Electric and Plug-in Hybrid Electric Vehicles”, Book, Springer, 2013. ISBN: 987-1-4614-7711-2.

[3] C. Chan, “The state of the art of electric and hybrid vehicles”, Journal, *Proceedings of the IEEE*, vol. 90, no. 2, pp. 247-275, 2002.

[4] K. Patil, S. Molla and T. Schulze, “Hybrid Vehicle Model Development using ASM-AMESim-Simscape Co-Simulation for Real-Time HIL Applications”, Journal, *SAE Technical Paper Series*, no. 2012-01-0932, 2012.

[5] H.-H. Braess und U. Seiffert, “Vieweg Handbuch Kraftfahrzeugtechnik”, Book, Vieweg+Teubner, Edition: 6, 2012, ISBN: 978-3-8348-8298-1.

[6] J. Hu, Y. Zhao and D. Qin, “Hardware-In-Loop Simulation of HEV System Based on CAN”, Journal, *Chinese Journal of Mechanical Engineering*, vol. 19, no. 3, pp. 300-304, 2008.

[7] BMW Group, “Virtual Energy Cells", Online Magzine, *dSPACE Magazine*, no. 1, pp. 6-11, 2010.

[8] C. Jefferson and R. Barnard, “Hybrid vehicle propulsion”, Book, WIT Press, 2002, ISBN: 978-1-85312-887-5.

[9] W. Li, “Research of Hybrid Electric Vehicle Control System and Energy Management Strategy”, Ph.D. dissertation, Shanghai Jiao Tong University, 2008.

[10] Robert Bosch Gmbh, “Bosch automotive electrics and automotive electronics”, Book, Springer Vieweg, 2014, ISBN: 978-3-658-01783-5.

[11] X. Yang, “Fieldbus Technology and its Application”, Book, Edition: 2, Tsinghua University Press, 2008, ISBN: 978-7-302-16993-2.

[12] D. Paret, “Multiplexed networks for embedded systems”, Book, Wiley & Sons, 2007, ISBN: 978-0768019384.

[13] S. Talbot and S. Ren, “Comparision of FieldBus Systems CAN, TTCAN, FlexRay and LIN in Passenger Vehicles”, Conference, in *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, 2009.

[14] L. Wang, “Design of Body Control Module in Vehicle Based on CAN/LIN Bus”, Master thesis, Hunan University, 2008.

[15] FlexRay Consortium, “FlexRay Communications System Protocol Specification Version 3.0.1”, Handbook, 2010.

[16] Fujitsu Microelectronics (Shanghai) Co.,Ltd., “Next Generation Car Network – FlexRay”, Handbook, 2006.

[17] M. Geimer, T. Krüger and P. Linsel, “Co-Simulation, gekoppelte Simulation oder Simulatorkopplung”, Journal, *O + P Zeitschrift für Fluidtechnik*, vol. 5011-12, pp. 572-576, 2006.

[18] G. Stettinger, M. Benedikt, N. Thek and J. Zehetner, “On the Difficulties of Real-time Co-simulation”, Conference, in *V International Conference on Computational Methods for Coupled Problems in Science and Engineering COUPLED PROBLEMS*, Ibiza, 2013.

_

[19] N. Thek, M. Benedikt and J. Zehetner, “Co-simulation Under Hard-real-time Conditions”, Conference, in *NAFEMS World Congress 2013*, Salzburg, 2013.

[20] G. Stettinger, J. Zehetner, M. Benedikt and N. Thek, “Extending Co-Simulation to the Real-Time Domain”, Journal, *SAE Technical Paper Series*, no. 2013-01-0421, 2013.

[21] M. Benedikt, D. Watzenig, J. Zehetner and A. Hofer, “NEPCE - A Nearly Energy-preserving Coupling Element for Weak-coupled Problems and Co-simulations”, Conference, in *V International Conference on Computational Methods for Coupled Problems in Science and Engineering COUPLED PROBLEMS*, Ibiza, 2013.

[22] W. Ecker, W. Müller and R. Domer, “Hardware-dependent Software”, Book, Springer, 2009, ISBN: 978-1-4020-9435-4.

[23] L. Cruz, “Jitter Analysis”, Technical report, FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO, 2011.

[24] SITime, “Clock Jitter Definitions and Measurement Methods”, Technical report, 2014.

[25] F. Proctor and W. Shackleford, “Real-time operating system timing jitter and its impact on motor control”, Journal, in *Sensors and Controls for Intelligent Manufacturing II*, vol. 4563, pp. 10-16, 2001.

[26] K. KOŁEK, “Windows7 x64 as real-time measurement and control platform”, Journal, *PRZEGLĄD ELEKTROTECHNICZNY*, vol. 89, no. 2, pp. 89-92, 2013.

[27] Y. Shan, “Research and Implementation of Key Technologies of Real-time Semi-physical Simulation Platform”, Master thesis, National University of Defense Technology, 2010.

[28] H. Xia, M. Lin, Q. Zeng and W. Xie, “Real- time Simulation System on the Ground for Satellite Attitude and Orbit Control Based on RTX”, Journal, *Chinese Journal of Computer Simulation*, vol. 23, no. 9, pp. 40-42, 2006.

[29] L. Wang, M. Lin, Q. Zeng and C. Wang, “Application of RTX in semi-physical simulation for laser-guided bomb”, Journal, *Chinese Journal of Infrared and Laser Engineering*, vol. 35, no. 1, pp. 78-81, 2006.

[30] Z. Jiang, “Design of Modeling and Real-time Simulation Software YH-RTSM Based on RTX”, Journal, *Chinese Journal of Computer Application*, vol. 30, no. 6, pp. 1635-1637, 2010.

[31] L. Xu, “Research and Implementation of Integrative Real-Time & Hardware-in-the-loop Simulation Platform Based on Simulink”, Master thesis, National University of Defense Technology, 2008.

[32] The Mathworks, “MATLAB – MathWorks”, 2017. [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed: 17- Dec- 2017].

[33] The Mathworks, “Simulink - Simulation and Model-Based Design”, 2017. [Online]. Available: https://www.mathworks.com/products/simulink.html. [Accessed: 17- Dec- 2017].

[34] M. Kogler, “Aufsetzen einer Software Testumgebung für reale Steuergeräte”, Master thesis, Graz University of Technology, 2016.

[35] D. Louarn-Pioch, “Function developement and test environment: AVLab, integrating Visu-IT ADD”, AVL internal documentation, 2014.

[36] AVL List GmbH, “Documenation of AVLab”, Handbook, 2017.

[37] AVL List GmbH, “Model.CONNECT^{TM} user manual”, Handbook, 2017.

[38] AVL List GmbH, “Job management system (JMS) users guide”, Handbook, 2017.

–

[39] Vector Informatik GmbH, “CANape - Calibrating ECUs optimally”, 2017. [Online]. Available: https://vector.com/vi_canape_en.html. [Accessed: 17- Dec- 2017].

[40] Vector Informatik GmbH, “CANape user manual”, Handbook, 2017.

[41] AVL List GmbH, “AVL CONCERTO 5™ - Calibration Tools - avl.com”, 2017. [Online]. Available: https://www.avl.com/-/avl-concerto-5-. [Accessed: 17- Dec- 2017].

[42] J. Weber, “P2 full hybrid E/E system specification”, AVL internal documentation, 2015.

[43] Continental, “HVCU target specification”, AVL internal documentation, 2015.

[44] The Mathworks, “How Acceleration Modes Work - MATLAB & Simulink - MathWorks United Kingdom”, 2017. [Online]. Available: https://www.mathworks.com/help/simulink/ug/how-the-acceleration-modes-work.html. [Accessed: 17- Dec- 2017].

[45] MRS, “Zeitrelais einstellbar abfallverzögert”, Datasheet, 2017.

[46] M. Hirz, “Module 1: Basics & Introduction into Mechatronics”, Lecture, Institute of Automitove Engineering, Graz University of Technology, Graz, 2017.

[47] AVL List GmbH, “Testbed.CONNECT™ - IODP Portfolio - avl.com”, 2017. [Online]. Available: https://www.avl.com/-/testbed-connect-. [Accessed: 17- Dec- 2017].

[48] Wikipedia, “New European Driving Cycle”, 2017. [Online]. Available: https://en.wikipedia.org/wiki/New_European_Driving_Cycle. [Accessed: 17- Dec- 2017].