FMI Application Guide (Appendix)

Ver. 2.0.2

February 7, 2026

Revision	Date	Main contents
1.0.0	10/01/2018	Ver1.0.0 released
1.0.1	11/01 2018	Chapter 6: Added 6.2.4, corrected a typo in 6.2.5 regarding minimum computation time.
2.0.0	10/31/2023	Ver. 2.0.0 released
2.0.1	06/01/2024	Ver. 2.0.1 released
2.0.2	02/07/2026	Ver. 2.0.2 released

Table of Contents

The following individuals, companies and organizations have participated and cooperated with the FMI Utilization and Deployment WG of the Automotive Control and Modeling Division Committee of the Society of Automotive Engineers of Japan in the preparation of this Guide.

Mutsuhito Eshima	IDAJ Corporation
Junichi Ichihara	AZAPA Corporation
Kazunari Moriya	AZAPA Corporation
Takayuki Sekisue	Ansys Japan K.K.
Takashi Iwagaya	Cybernet Systems, Inc.
Martin Egginton	Siemens AG
Yosuke Ogata	Siemens AG
Yuuichiro Abe	Dassault Systèmes K.K.
VIRY, Guillaume	Dassault Systèmes K.K.
Makoto Koekiba	Chuo Zuken K.K.
Katsuya Tsuzuki	dSPACE Japan Corporation
Takashi Yoshimatsu	dSPACE Japan Corporation
Yumi Uchida	Dentsu Research Institute Inc.
Keiichi Ueyama	Denso Corporation
Shuya Miwa	Denso Corporation
Dai Araki	Toshiba Digital Solutions Corporation
Yutaka Hirano	HIRANO Research Lab.
Yasufumi Saruki	HEXAGON Corporation
Nobuyuki Hirai	HEXAGON Corporation
Motoaki Muto	Nissan Motor Co.
Haruki Saito	Nissan Motor Co.
Hideaki Jinno	Honda R&D Co.
Shingo Haketa	Honda R&D Co.
Riichi Nagao	PonoSHIP K.K.
Daisuke Akasaka	The MathWorks GK
Akio Takashima	The MathWorks GK
Shun Endo	Mazda Motor Corporation
Ken Komori	Mazda Motor Corporation
Hiroaki Takeuchi	Mazda Motor Corporation
Yuji Sato	Mitsubishi Space Software Corporation
GAO, Rui	Modelon Corporation

(Alphabetical order by company/organization name)

We would also like to thank the following individuals, companies and organizations for their cooperation in preparing this guide.

Seiji Ishikawa	ETAS Corporation
Kensuke Shibuya	Nagoya University
Koichi Shigematsu	Nagoya University
Yutaka Funabashi	Renesas Electronics Corporation

3V-SG (Study Group for Control Verification Using Virtual Methods)

(Alphabetical order by company/organization name)

Chapter 1: Introduction to FMI-compliant tools

This chapter introduces FMI-related functions in typical tools.

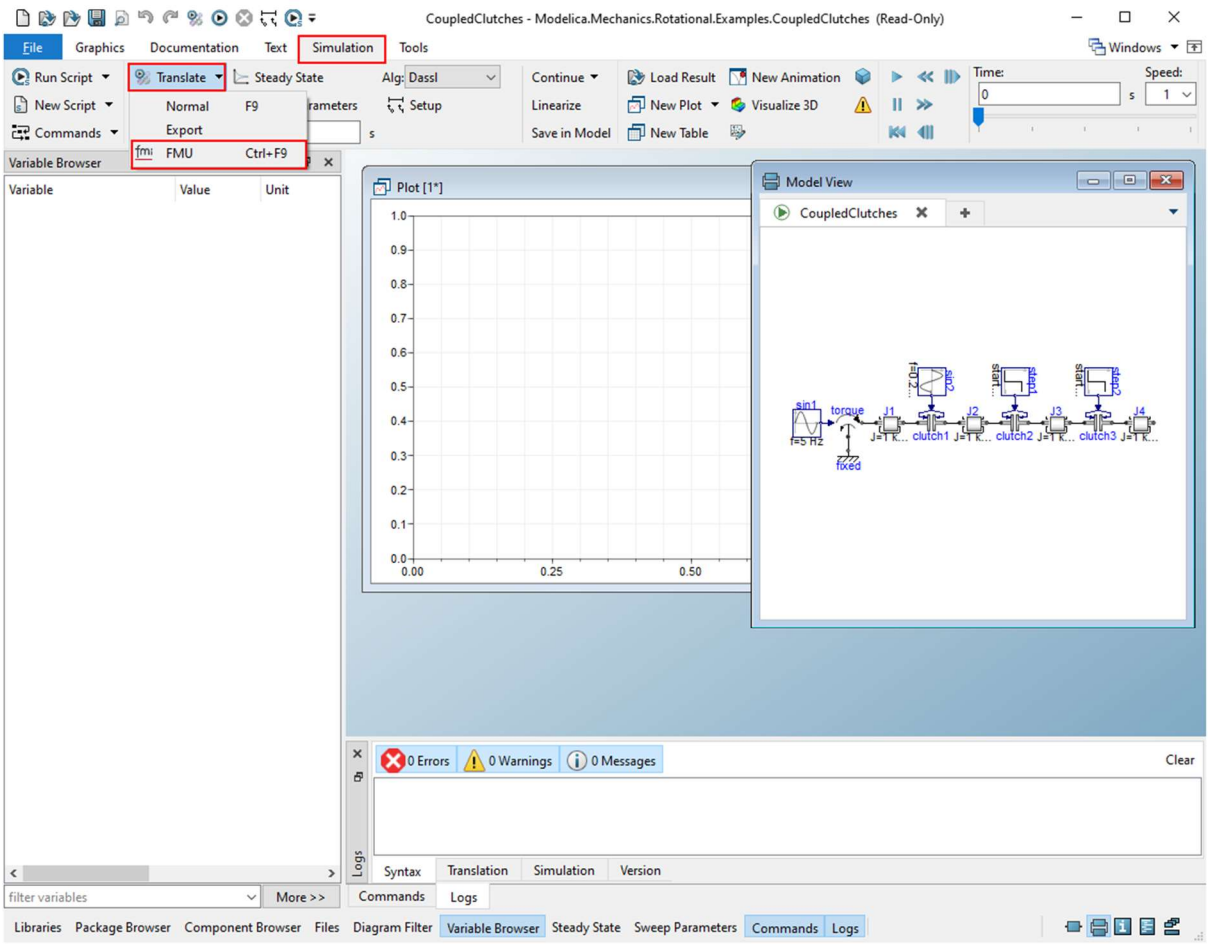
1.1. Dassault Systèmes Dymola

Dymola is a simulation tool using the Modelica language from Dassault Systemes (Dassault Systèmes) Dymola follows the FMI standard and allows the creation, integration and simulation of FMI-compliant models.

- Supports FMI 1.0, 2.0, and 3.0 (beta).
- Supports Co-Simulation and Model Exchange.
- Windows and Linux are supported.

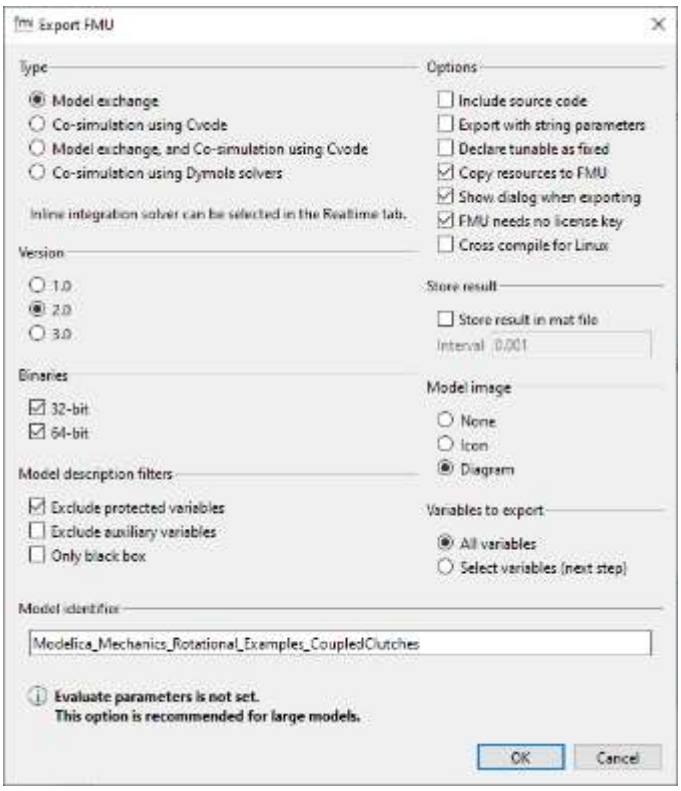
1.1.1. Creating FMUs

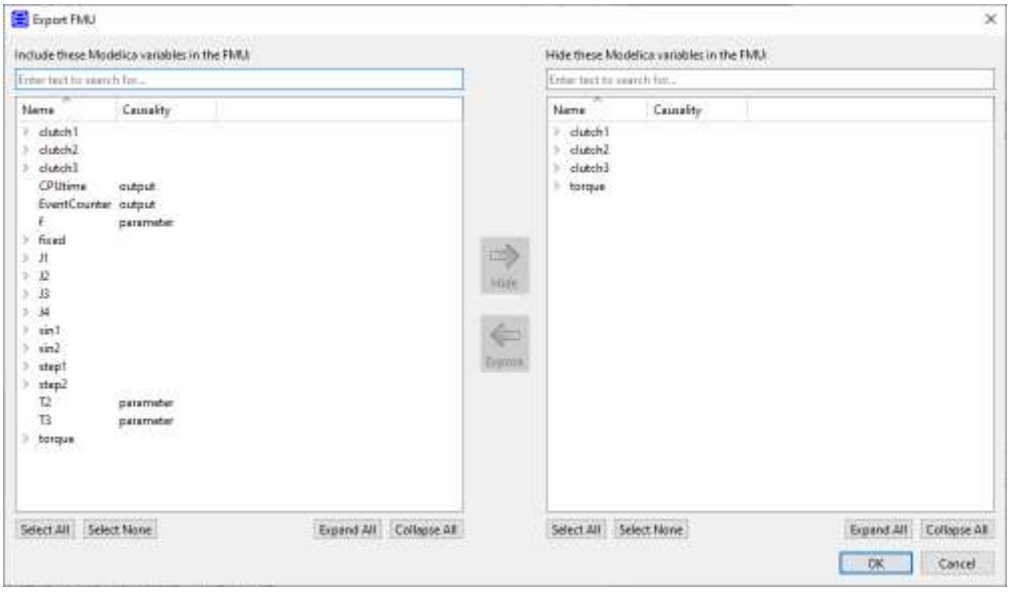
If you want to generate an FMU from the model, go to the Simulation tab and select Translate. Then select FMU.



On the next screen, you will see options for FMU generation. Of these, choose the type of FMU (Model Exchange only, Co-simulation only, or Hybrid FMU), standard version, platform, etc.If you choose a co-simulation FMU, the solver settings will be set to the state of the solver at the time the

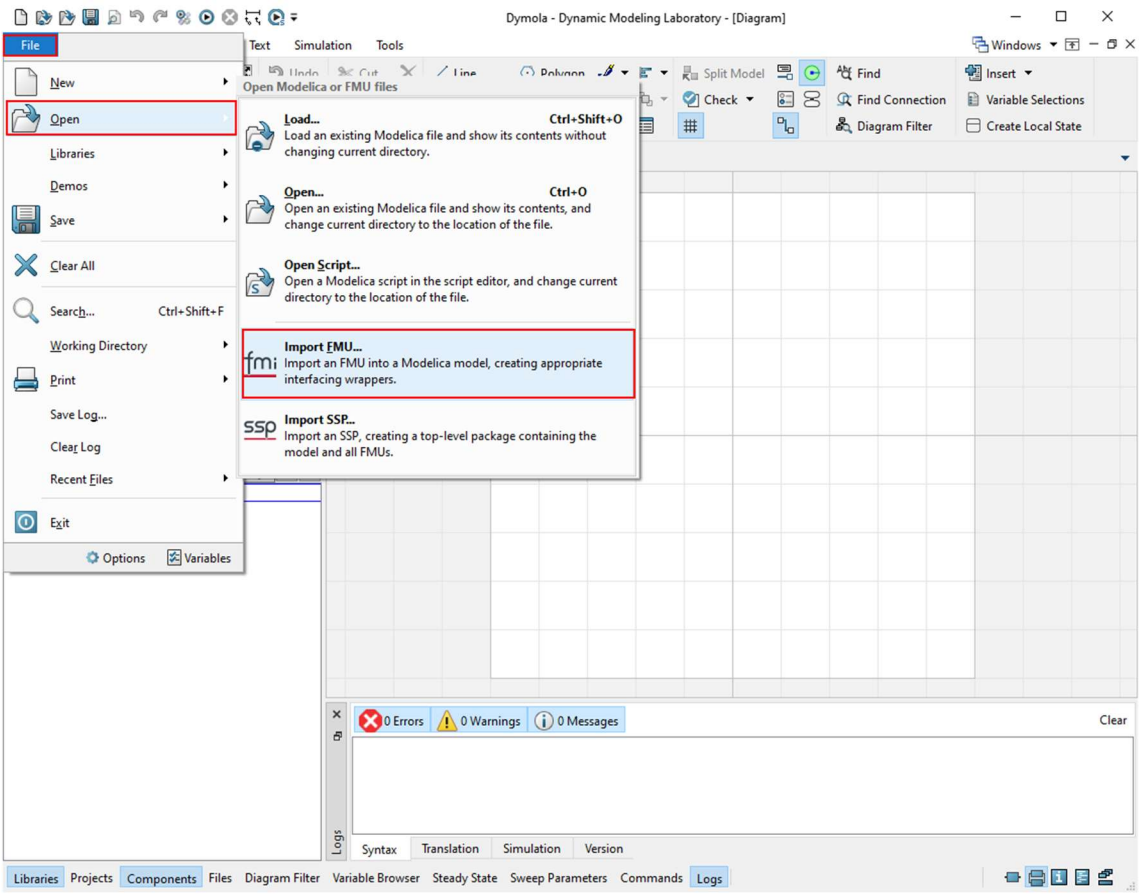
command is executed. The solver settings will be the same as they were when the command was executed. You can also decide which variables to exclude, if necessary (for example, if you only want to expose I/O and parameters, choose the Black-Box FMU). If you want to select variables in more detail, check Select variables. Then the next screen will allow you to select variables.



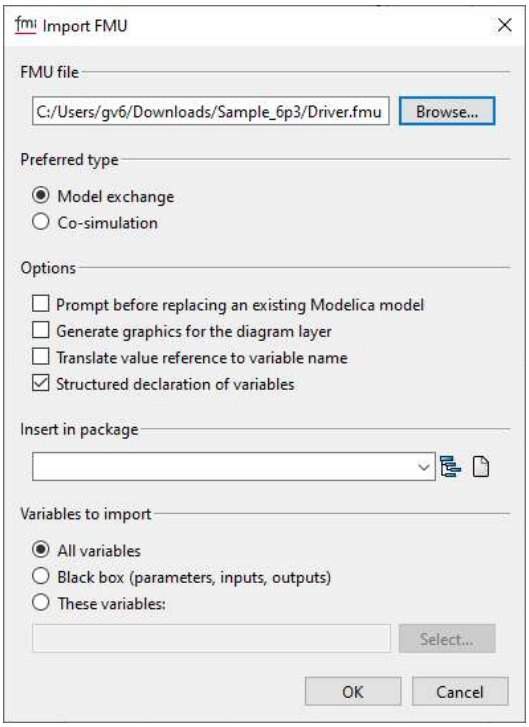


1.1.2. FMU import and execution

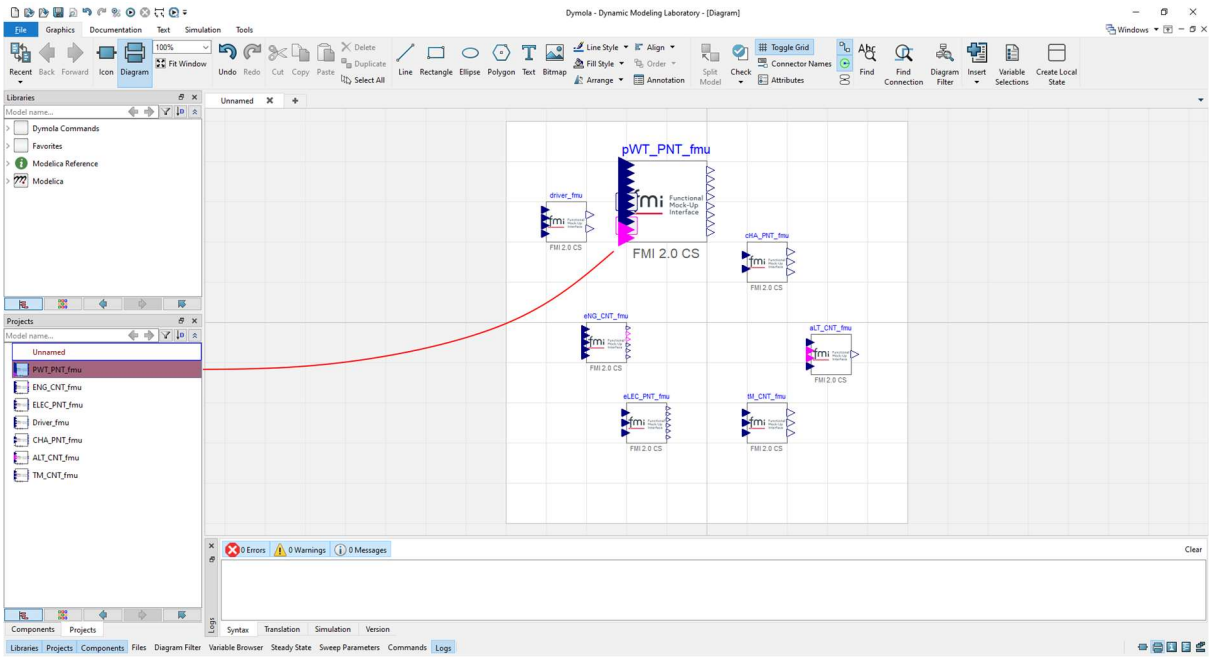
If you want to import an FMU, select the Open command in the File menu and choose Import FMU. You can also use Drag & Drop in the middle of the screen.



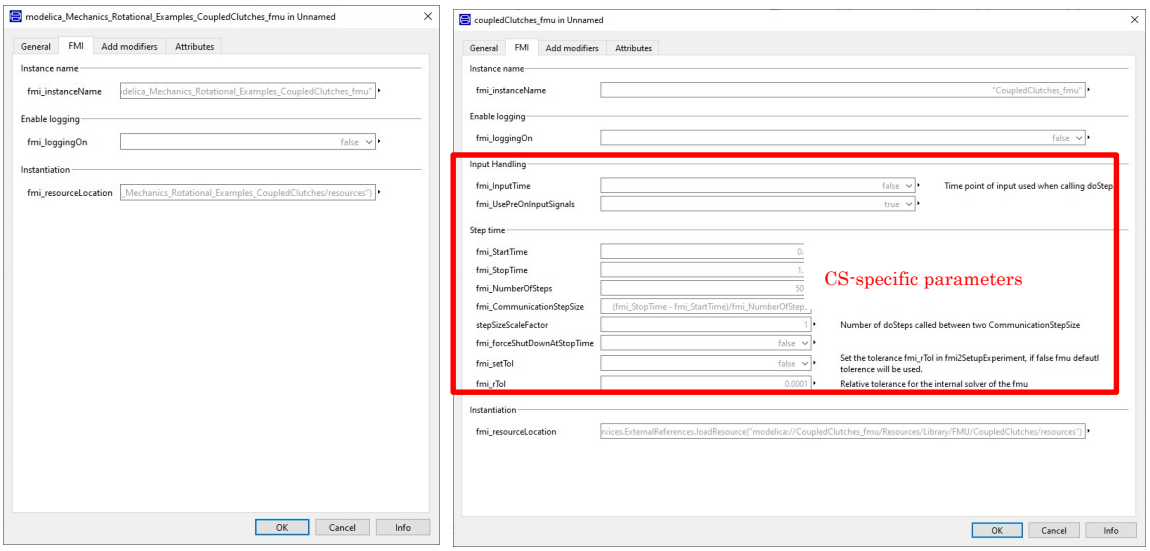
Specify the path to the file and choose the preferred mode of use for the hybrid FMU. At that stage, you can decide in detail which variables to expose.



To run FMU, move the imported model to the main screen.



Double-clicking on a component brings up a screen where you can set parameters. Among them, there is a tab dedicated to FMI, where you can specify FMU logging, instance name, etc. In the case of co-simulation (screenshot on the right), there are also start time, end time, and synchronization steps.

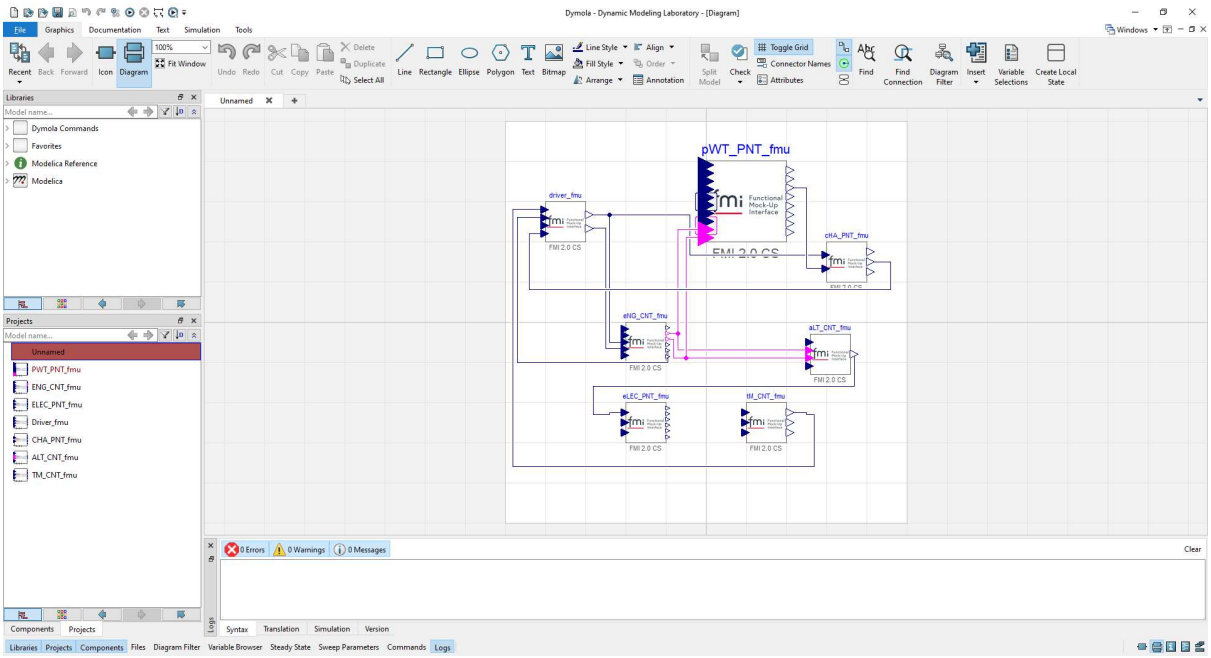


Parameters of the Model Exchange FMU *Parameters of Co-Simulation FMU*

The imported FMU can be connected to other FMUs and Modelica components and run like a regular Dymola simulation to display and analyze results.

In case of co-simulation, make sure that the I/O synchronization steps for each FMU are set correctly.

A model that includes multiple FMUs looks like the one below (with component connections not yet completed).


Dymola - Dynamic Modeling Laboratory - [Diagram]
pWT_PNT_fmu
driver_fmu
cHA_PNT_fmu
eNG_CNT_fmu
aLT_CNT_fmu
eLEC_PNT_fmu
tM_CNT_fmu
FMI 2.0 CS
Libraries
Projects
Dymola Commands
Favorites
Modelica Reference
Modelica
Unnamed
PWT_PNT_fmu
ENG_CNT_fmu
ELEC_PNT_fmu
Driver_fmu
CHA_PNT_fmu
ALT_CNT_fmu
TM_CNT_fmu
0 Errors
0 Warnings
0 Messages
Clear
Components
Projects
Syntax
Translation
Simulation
Version
Libraries
Projects
Components
Files
Diagram Filter
Variable Browser
Steady State
Sweep Parameters
Commands
Logs

1.2. Ansys Twin Builder

Features of Twin Builder (2023.R2) are shown in table 1.2.1.

When creating an FMU, the functions and creation methods differ depending on whether you create an FMU from a model using only the Modelica library, or from a model created by a sub-sheet structure in combination with Twin Builder's own library (SML).

FMI creation function (Modelica)	
FMI Version	1.0, 2.0
Form	Model Exchange, Co-Simulation
OS	Windows 64, 32
Execution License	unnecessary
FMI creation function (sub-sheet format)	
FMI Version	2.0
Form	Co-Simulation
OS	Windows 64, Linux 64 (Ubuntu, RedHat)
Execution License	Not required (can be switched to required)
FMI read function	
FMI Version	1.0, 2.0
Form	Model Exchange, Co-Simulation
OS	Windows 64
Execution License	Depends on the creator

Table1.2.1 List of Twin Builder FMI Interface Functions

1.2.1. FMU creation (Modelica)

This section describes the procedure for creating an FMU using only models created with the Modelica editor.

A) Modelica model creation and compilation

① Right-click on Definitions/Models in the project tree and select "Add Definition" to specify the name of the model to be created and open the Modelica editor.

②The Modelica library tree will appear on the right side of the screen, and you can drag and drop parts, connect wires, etc. to build the target model.

③ Execute compilation from the ribbon [Compile & Update Project].

④ Close the Modelica Editor by pressing the "X" button directly below the upper left corner of the screen.

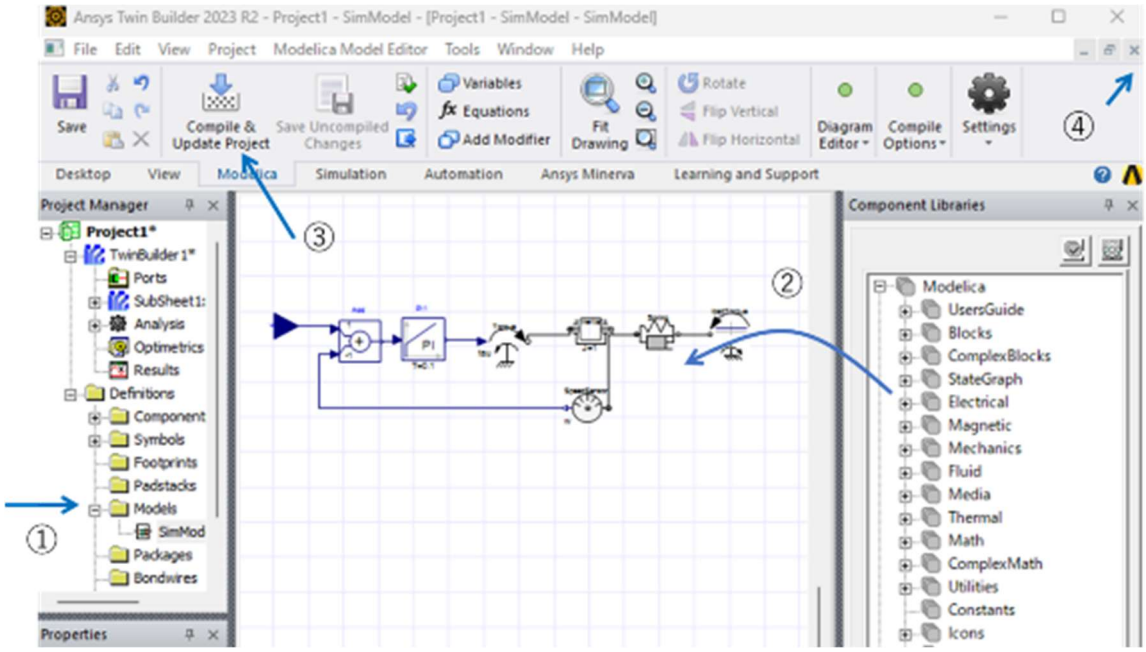


Figure1.2.1　Modelica model editing screen

B) Creation of FMU

① Right-click the Modelica model created from the project tree Definitions/Models and select "Export as FMU".

②Select FMU file name, type, target OS, etc., and click [Export].

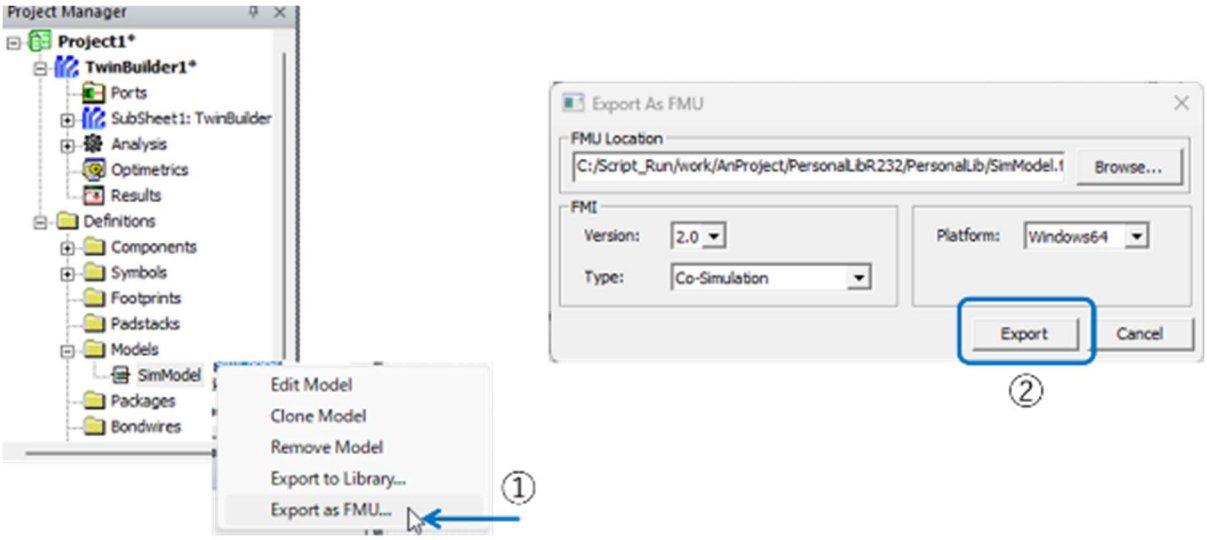


Figure1.2.2　FMU creation screen from Modelica model

These procedures are for generating an FMU from a model consisting only of the Modelica library. The created FMU model name will be the Modelica model name specified at A)-①.

1.2.2.　Creation of FMU (sub-sheet format)

This section describes the procedure for creating FMUs by creating a subsheet of Twin Builder's proprietary library models (SML), Modelica and VHDL-AMS models, C language models, etc., and

then creating FMUs the hierarchy of these models. Here, SML models mainly refer to the parts included in the library trees BasicElements and Multiphysics.

A) Preparation of subsheet and connection terminals

① Clicking on the ribbon [Subcircuit] will generate a sub-sheet (SubSheet1) under the top-level sheet (TwinBuilder1).

② Click the Interface port icon on the ribbon, and the terminal symbol will appear on the mouse cursor.

③ Double-click the placed terminal to display the dialog shown on the right to modify the terminal name and attributes. The following is a list of the items that can be done with regard to

Domain : Quantity (input/output signal) Parametric (fixed value parameter)

Available only in Type : real

Direction : in (input signal) out (output signal)

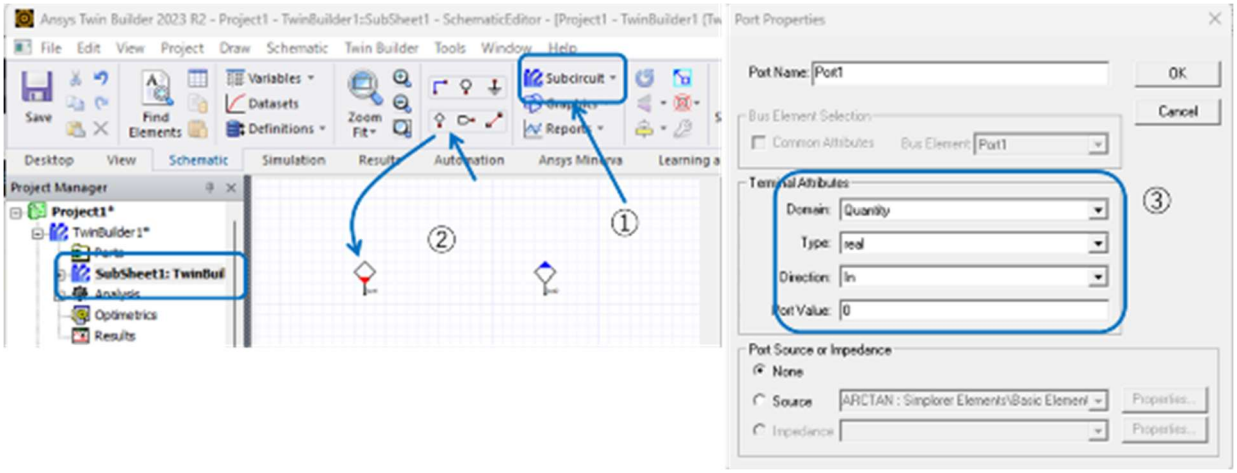


Figure1.2.3 Creating a sub-sheet

B) Creation of sub-sheet internal configuration

① Place components from the library tree displayed on the left side and connect them to the terminals you have created.

Components can also include Modelica and loaded FMUs. These are located in the bottom [Project Components] of the library tree.

② When the model is complete, right-click on a blank area of the screen and select "Pop Up" to return to the top-level hierarchy.

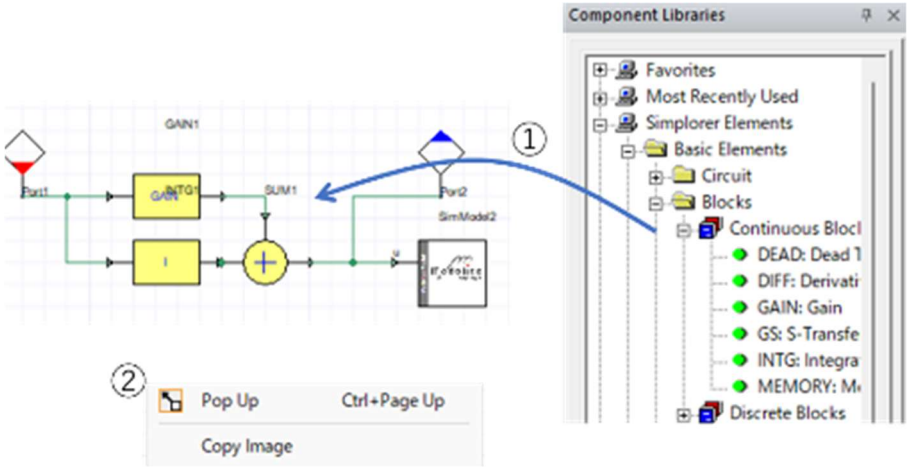


Figure1.2.4　Creation of sub-sheet interior

C) Creation of FMU

① Sub-sheets are created in the lower left corner of the screen. Right-click on the model and select [Compile as co-simulation FMU] to open the dialog box in the upper right corner of the figure.

② Select [Auto Detect] to automatically recognize the dlls of the components used internally. In addition, to create an FMU that also supports Linux64, you must separately install Twin Builder Linux Redistributable (free of charge) and specify its storage location.

③ After compilation, the FMU model (FMUModel_TwinBuilder2) is saved under Definitions/Models in the project tree. Right-click on it and select [Export as FMU].

④ Only FMI 2.0 Co-Simulation can be output, and it is output by [Export].

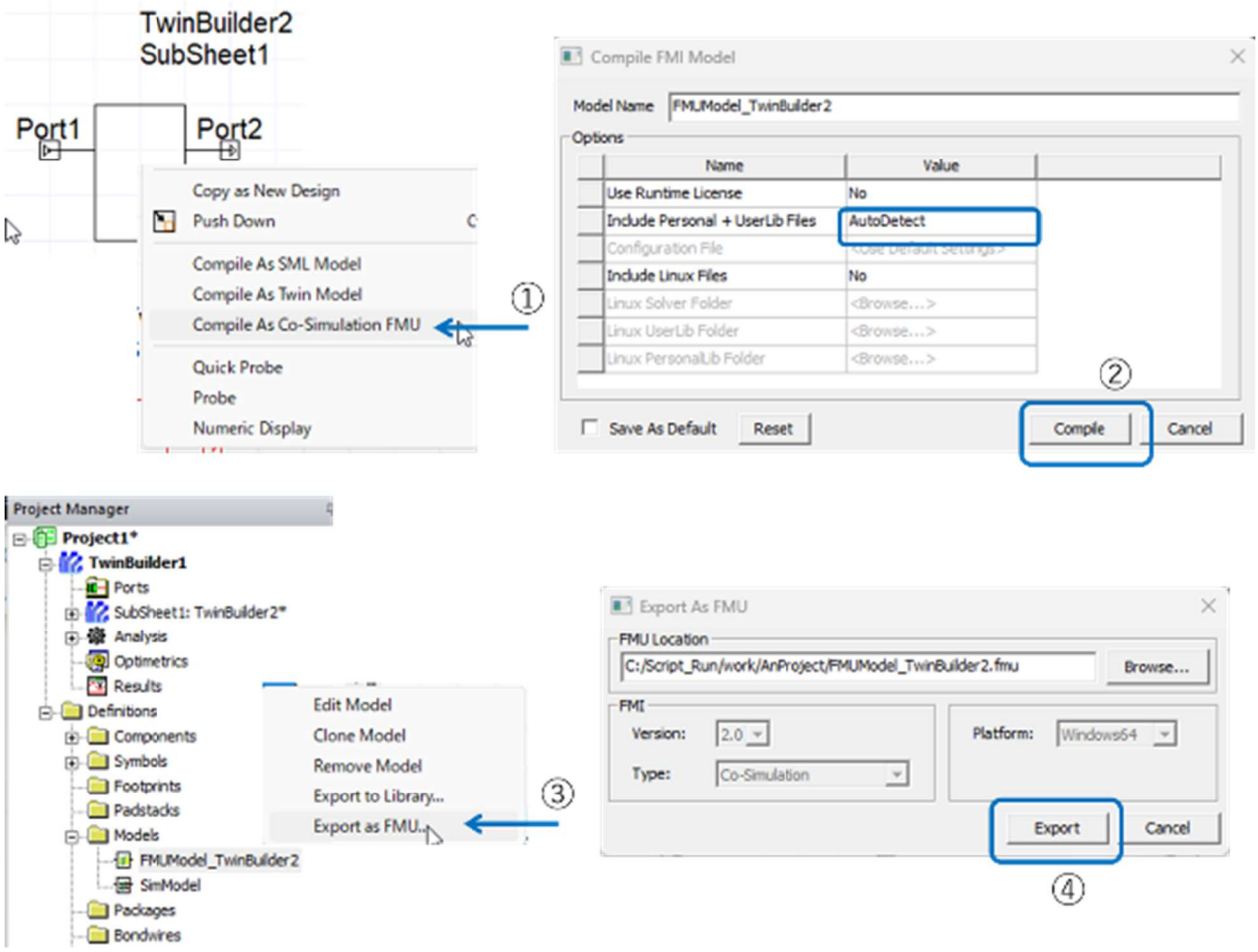


Figure1.2.5　FMU output in sub-sheet format

These procedures are for generating an FMU from a model created in the subsheet format, and the name of the FMU model created is determined when compiling at C)-②.

1.2.3.　Loading FMUs

Twin Builder is only available for FMUs that include the Windows64 architecture.

A) Reading FMU files

From the top menu Twin Builder/ Add Component/ Add FMU Component, select any FMU file.

When the dialog box shown on the left side of fig.1.2.6, select "Import" and the FMU model will be placed on the cursor. The tree displayed in the dialog is a list of parameters for which you want to display results or change values, but you can freely add or delete parameters later.

② Right-click on the placed FMU component and select "Edit Model" to display the dialog box shown on the right.

③ Selecting another FMU file with the [Replace] button allows you to change the model entity; use this to switch between Model Exchange ⟷ Co-Simulation models or to update an FMU model that has been modified.

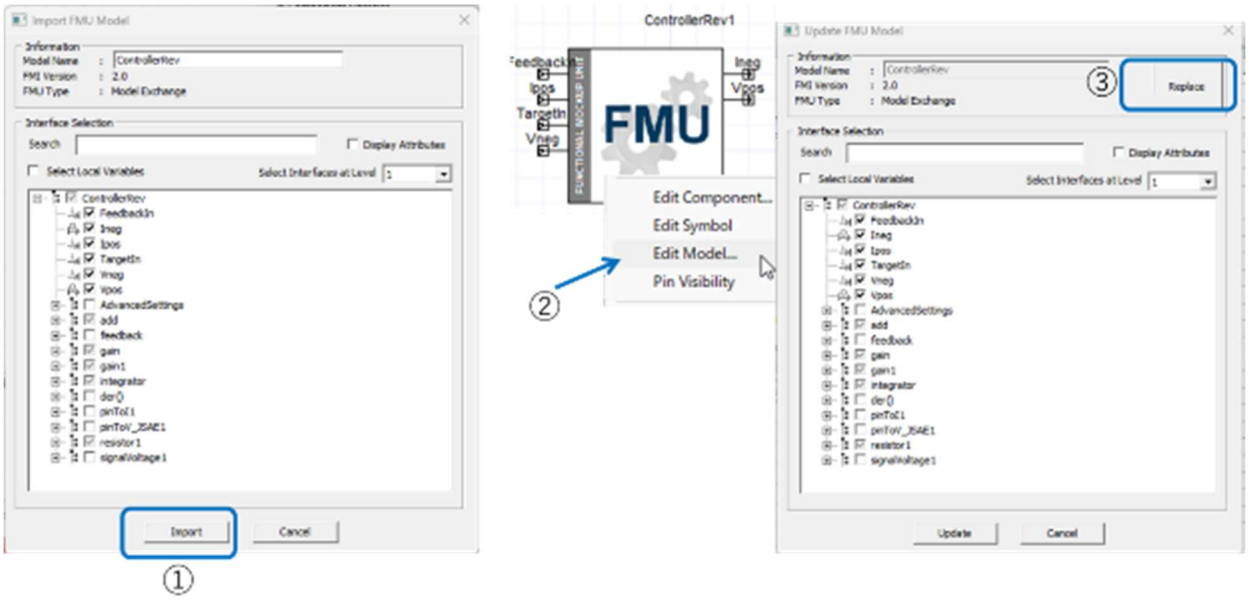


Figure1.2.6　Loading an FMU

1.2.4.　Others

Modelica Operations

<install-dir>¥Help¥Twin Builder¥GSG¥ModelicaTutorial.pdf

Operations on sub-sheets

<install-dir>¥Help¥Twin Builder¥Twin Builder.pdf Chapter 11 (Adding a Subcircuit to a Twin Builder Design)

Operation details regarding FMU

<install-dir>¥Help¥Twin Builder¥Twin Builder.pdf Chapter 6 (Exporting a Model as an FMU), Chapter 11 (Creating and Exporting Co-Simulation FMU Models)

Operation details for FMU

1.3. MathWorks Simulink

Simulink provided by MathWorks supports FMU import/export. The followingTable1.3.1 below summarizes the support status in MATLAB version R2023b.

Table1.3.1 FMI Support Status in Simulink (as of R2023b)

FMU Export	
FMI Version	2.0, 3.0
Form	Co-Simulation (CS)
To export Required License	MATLAB, Simulink, MATLAB Compiler, Simulink Compiler
Remarks	• FMU export requires separate Simulink Compiler*1 • FMI 3.0 CS support from R2023b*3
FMU Import	
FMI Version	1.0, 2.0, 3.0
Form	Co-Simulation (CS), Model Exchange (ME)
To import Required License	MATLAB, Simulink
Remarks	• Standard Simulink functionality (FMU block*2) supports FMU import • FMI3.0 CS/ME support from R2023b*3

*1. Simulink Compiler is an add-on product released for R2020a.

*2. FMU block is a Simulink block for FMU import introduced in R2017b.

*3 Some FMI 3.0 specifications are not supported as of R2023b. Please refer to the product documentation for details. Also, to export FMU from R2023b, you need to download and install the FMU Builder for Simulink support package (free of charge).

1.3.1. FMU export

Simulink Compiler is required to export FMUs from Simulink models. Below is a description of how to export FMUs using Simulink Compiler. Note that as of R2023b, only Simulink models running in Rapid Accelerator Mode simulation mode are supported for FMU export. Starting with R2023b, variable step solvers are also supported for generating FMI2.0 CS (however, MATLAB Runtime is required for the environment in which the FMU is run). Please refer to the product documentation for detailed limitations.

First, prepare a Simulink model to generate the FMU. The followingFigure1.3.1 From the Simulation tab at the top of the editor, select Save > Standalone FMU.

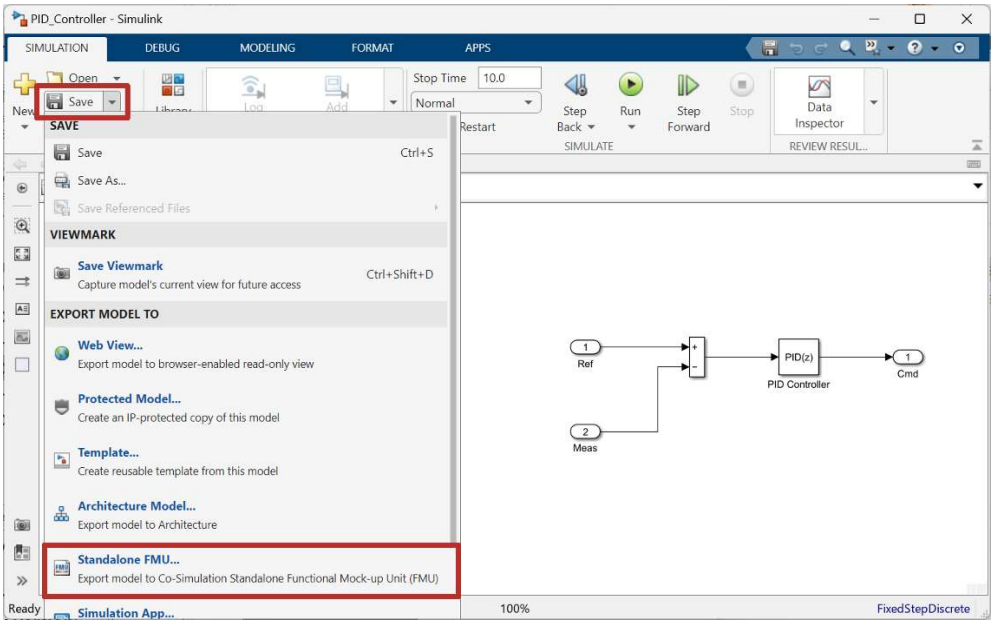


Figure1.3.1　FMU export from Simulink model

Then,Figure1.3.2 After setting the FMI version, parameters, input/output, etc., finally press the "Create" button at the bottom of the screen to generate the FMU.

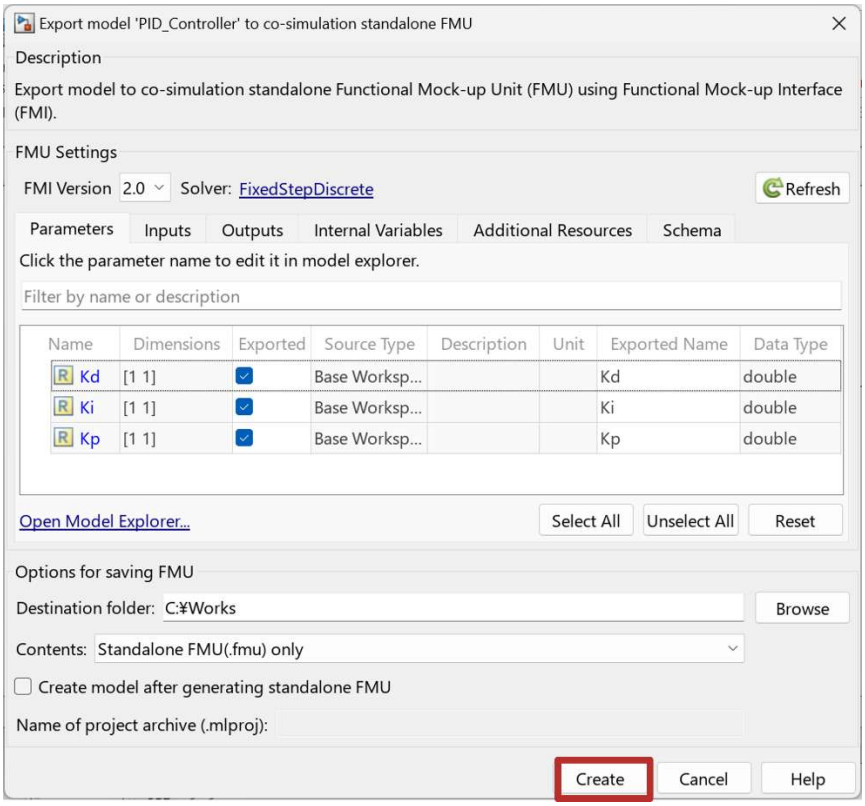


Figure1.3.2　FMU Export Settings Screen

In addition to FMU generation by GUI operation as described above, FMU generation by MATLAB command line using the exportToFMU2CS function (introduced in R2020a) or exportToFMU function (introduced in R2023b) is also possible.

1.3.2. FMU import and execution

Simulink provides "FMU blocks" for importing FMUs, which are stored in [Simulink Extras/FMU Import] in the Simulink library. The following sections describe how to import FMU and run simulations. Note that FMU import is a standard feature of Simulink and does not require Simulink Compiler.

Figure1.3.3 Place the FMU block on the Simulink editor campus as shown in Figure 1.3.3, with the FMU file not yet selected, it will appear in red as Unspecified FMU Import. Double-click the block to display the Select FMU File screen and select the FMU file you wish to import from the folder (in this example, ELEC_PNT.fmu).

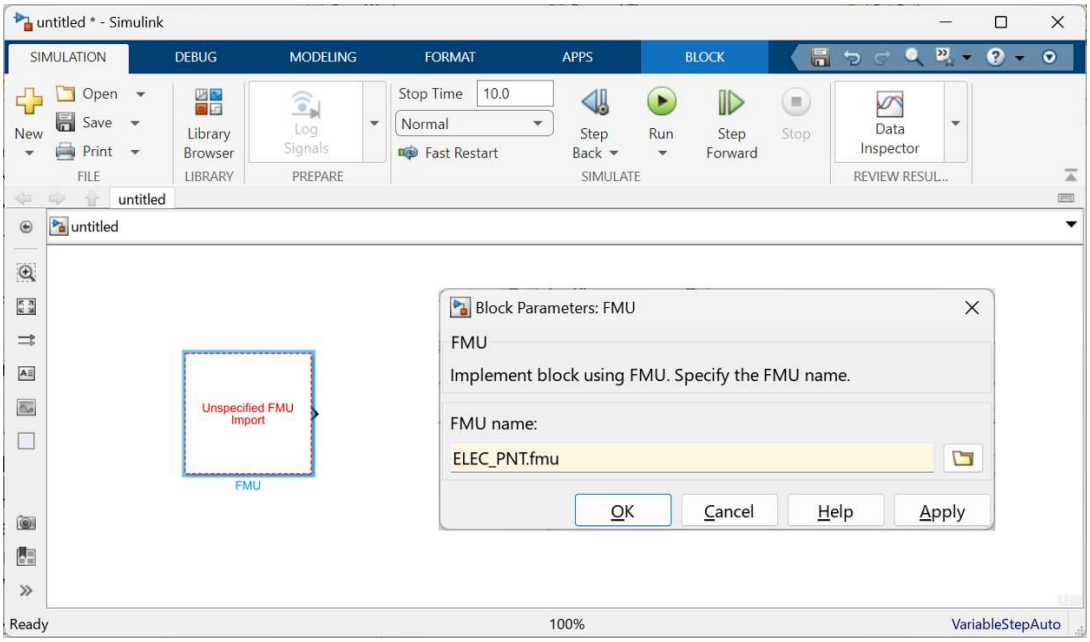


Figure1.3.3 Importing an FMU into a Simulink model

Selecting an FMU file,Figure1.3.4 The block display changes as shown in Figure 1.3.4, and input/output ports appear. Double-click the block to display the parameter setting screen and make the necessary settings.

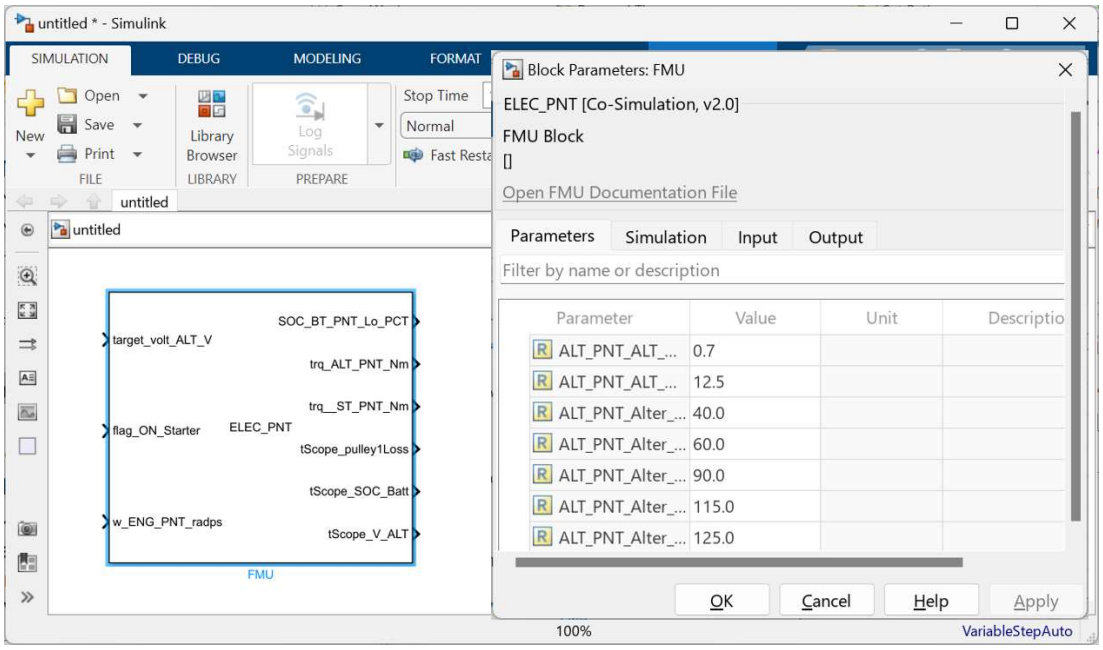


Figure1.3.4　FMU Block

The FMU block can be wired with signal lines to other Simulink blocks. Once the model is complete, Press the Simulation Run button (green round arrow button) on the Simulation tab at the top of the editor to start the simulation (Figure 1.3.5).

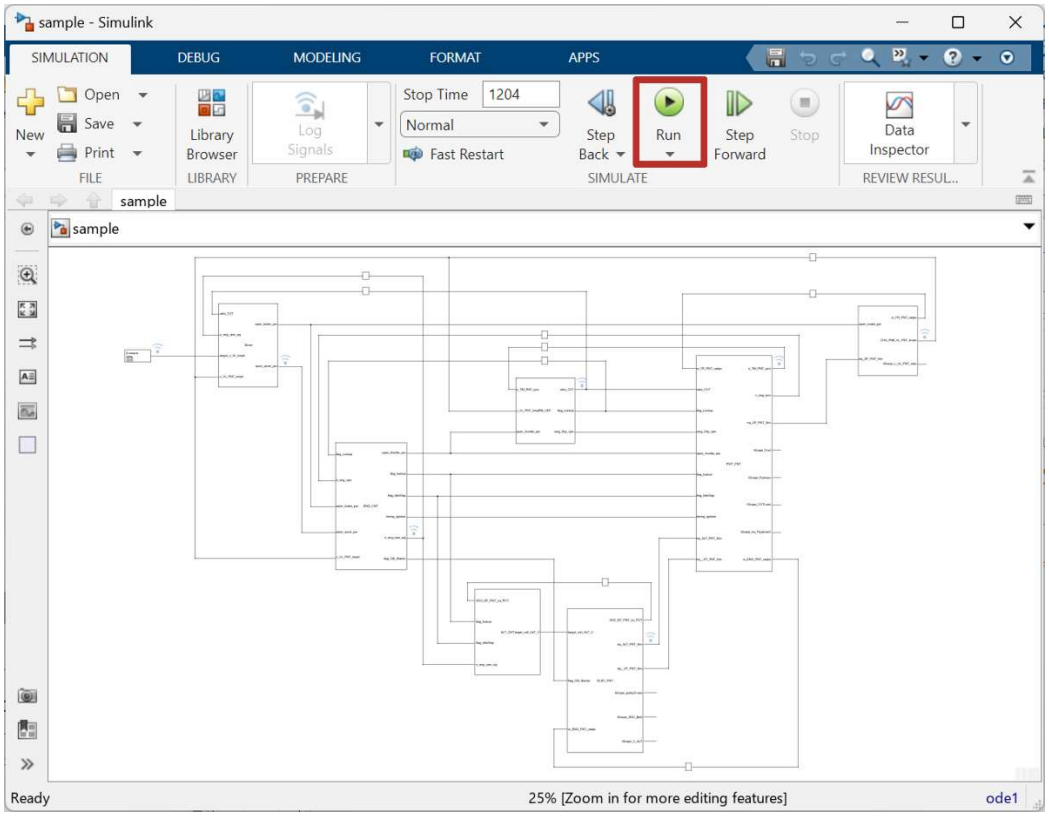


Figure1.3.5　Simulation of Simulink model with integrated FMU

1.4. Altair Twin Activate

Twin Activate is an integrated platform that allows multi-physics analysis for comprehensive model-based development. It has a block diagram library for control, and physical components can be hybrid modeled by leveraging Modelica libraries and other electrical, electronic, and hydraulic system libraries. It also allows for model integration with other Altair products such as MotionSolve and Flux.

Twin Activate supports FMI import/export, and table 1.4.1 below shows the support status in Ver. 2023. All functions that were supported for import/export in FMI 2.0 are supported in FMI 3.0, as well. In addition, you can export romAI blocks as an inline FMU using FMI 2.0 and FMI 3.0 export.

Table 1.4.1 List of FMI Interface Functions of TwinActivate

FMU Export	
FMI Version	2.0, 3.0
Form	Co-Simulation (CS), Model Exchange (ME)
OS	Windows 64, Linux 64
Execution License	Not required *1
remarks	• FMI3.0 CS/ME support from Ver. 2022.3
FMU Import	
FMI Version	1.0, 2.0, 3.0
form (something takes)	Co-Simulation (CS), Model Exchange (ME)
OS	Windows 64, Linux 64
Execution License	Depends on the creator
remarks	• Supports FMU import with standard Twin Activate functionality (FMU block) • FMI3.0 CS/ME support from Ver. 2022.3

*1. romAI license is required if the FMU contains romAI blocks.

1.4.1. Creating FMUs

① Create a model in Twin Activate. If you want to create input/output ports when creating an FMU, you need to place "Input" and "Output" blocks from "Ports" of the Activate library in the Palette Browser.

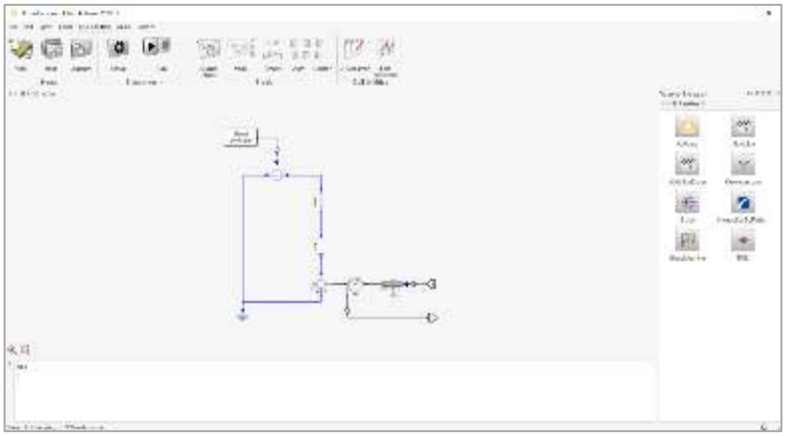


Figure 1.4 .1 Twin Activate model

② Next, use the "Super Block" function to group the model blocks that you want to be integrated into the FMU. Select the model excluding ports, right-click, and select "Super Block" to create it.

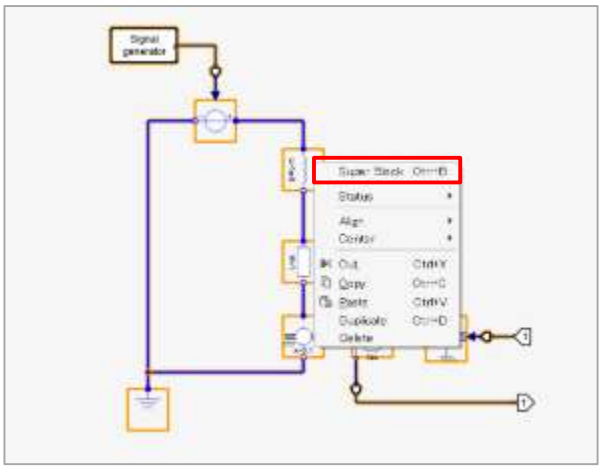


Figure 1.4.2 Super Block creation

③ Double-click on the superblock to expand it, then double-click on the Input port block within to open the Properties dialog. In the dialog, check the "Time dependency" checkbox. When this item is not selected, "Force inputs to always active" is checked in the dialog in Figure 1.4.5, no error will occur.

Figure1.4.3 Input port block settings

If you want to hide the parameters of the model when creating the FMU, perform the hiding process on the superblock parameters at this stage.

④ Select the created superblock and choose "Code Generation and Export" under "Tools".

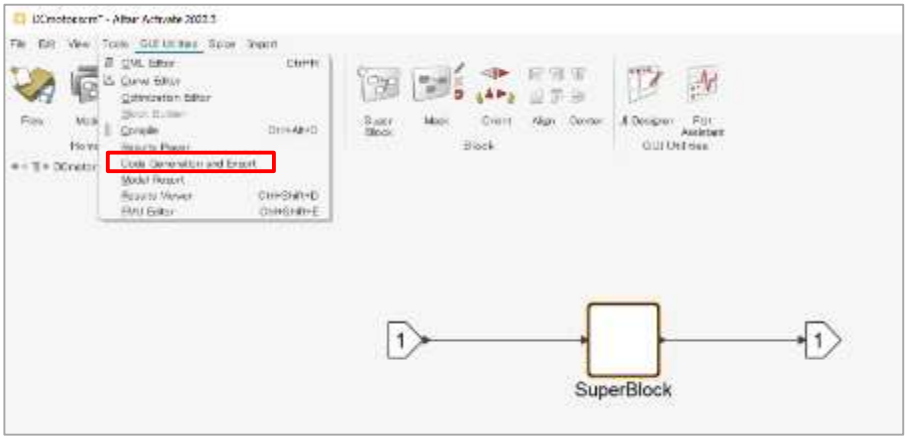


Figure1.4.4 Selecting Code Generation and Export

If "Target" is set to "FMU" in this dialog, "FMU type", "FMU version", and other options related to the generation will be displayed to select according to your needs.

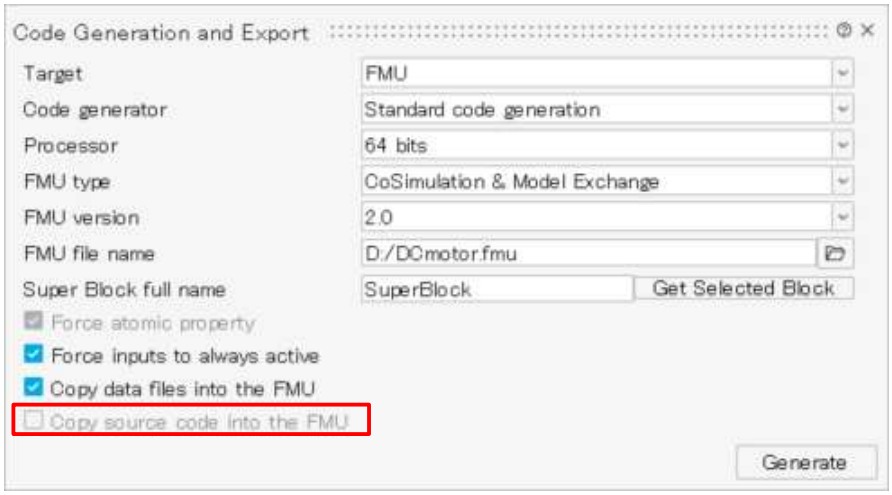


Figure1.4.5 FMU generation options

"Model Exchange", "CoSimulation", and "CoSimulation & Model Exchange" are available as

"FMU type", and 2.0 and 3.0 are available as "FMU version".
For "Code generator" select "Standard code generation" if you want to use the Activate block or a hybrid model with other Modelica blocks on a Windows/Linux platform. For usage of the Activate model within a hardware environment, select "Inlined code generation". However, there are blocks that are currently not supported by "Inlined code generation". Now, you can copy the source code into the FMU by checking the "Copy source code into the FMU" checkbox in the lower left corner.

⑤ After selecting the appropriate items, click the "Generate" button to generate the FMU in the directory specified in the "FMU file name".

1.4.2. FMU import and execution

① Select the "FMU" block from "CoSimulation" of the Activate library in the Palette Browser and place it in the modeling window.

Figure 1.4.6 FMU block placement

② Double-click on the FMU block and specify the corresponding FMU in the "FMU file name" in the "General Parameters" tab of the properties dialog. FMU input/output port and parameter information will be loaded, automatically.

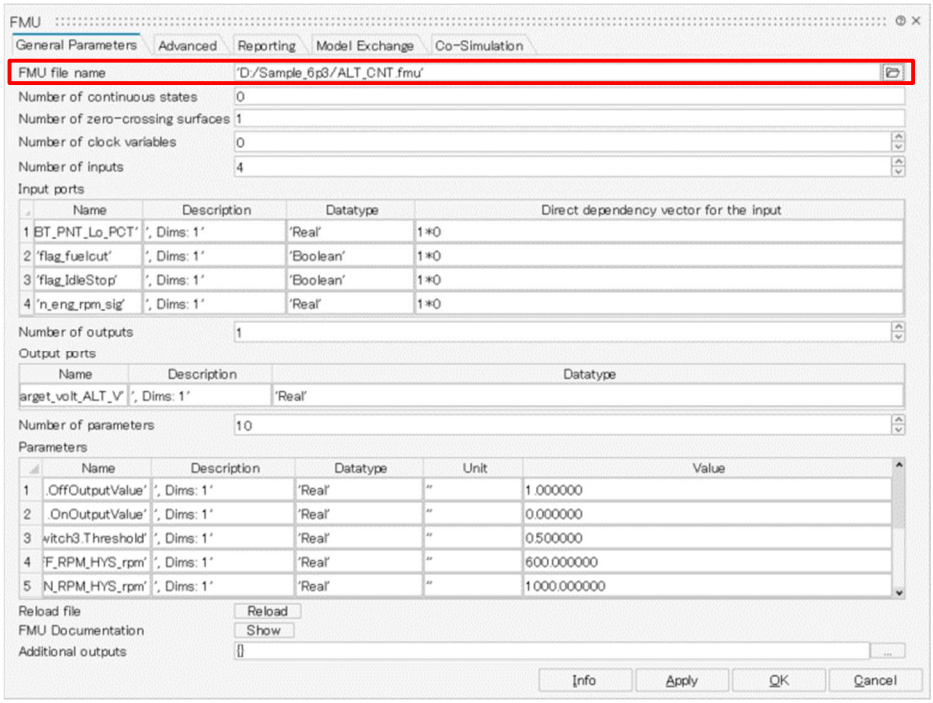


Figure1.4.7 Importing FMUs

③ The "Advanced" tab allows you to specify various additional parameters. If you are importing a hybrid FMU that includes both - Model Exchange and Co-Simulation - you can run it as Model Exchange by checking the "Run as Model Exchange If both FMU types are provided" checkbox within this tab. When importing a Co-Simulation FMU, you can set the communication interval for Co-Simulation in the "Preferred fixed communication step size" field within the "Co-Simulation" tab. After clicking "OK" in the dialog, the corresponding FMU will be placed as a block in the modeling window.

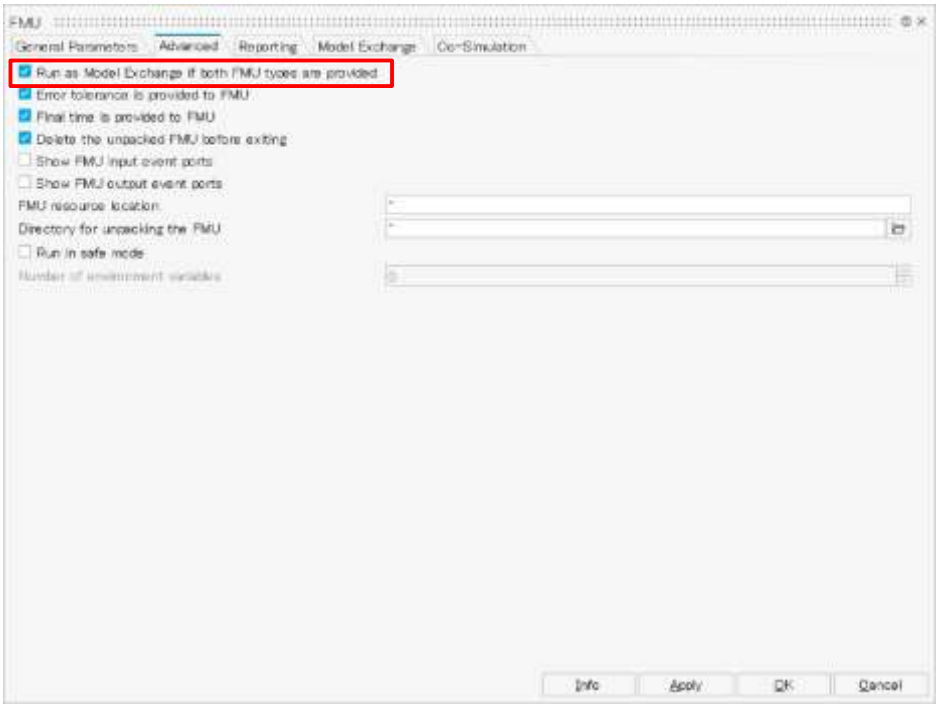


Figure 1.4.8 Advanced tab

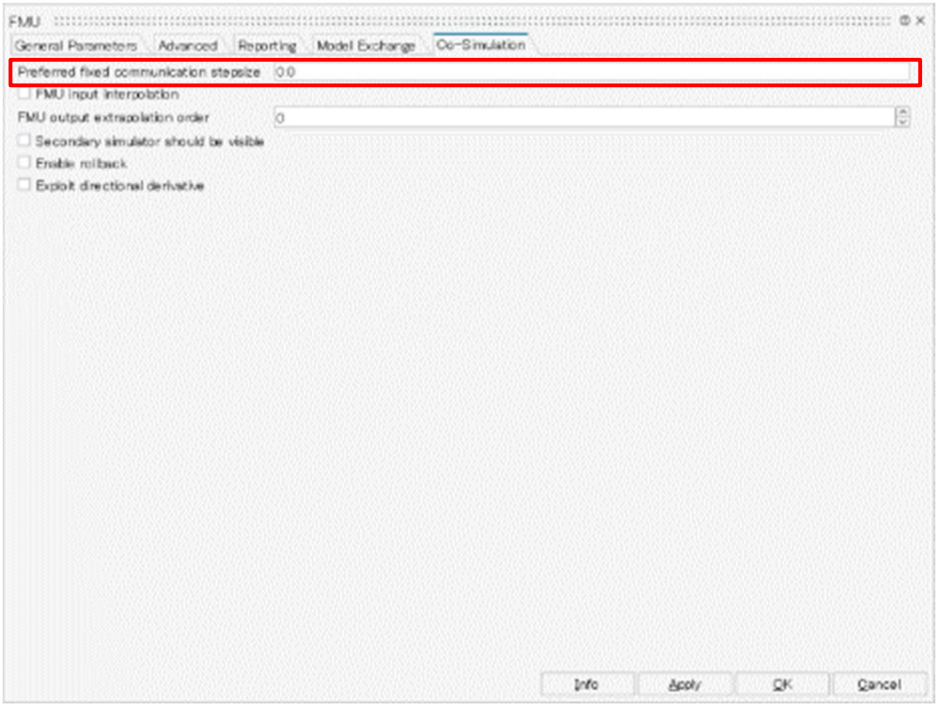


Figure1.4 .9 Co-Simulation Tab

④ Import all necessary FMUs as described above and connect them appropriately as signal names are displayed in the ports of the FMU block. Figure 1.4.10 below shows an example of a model. with multiple FMUs.

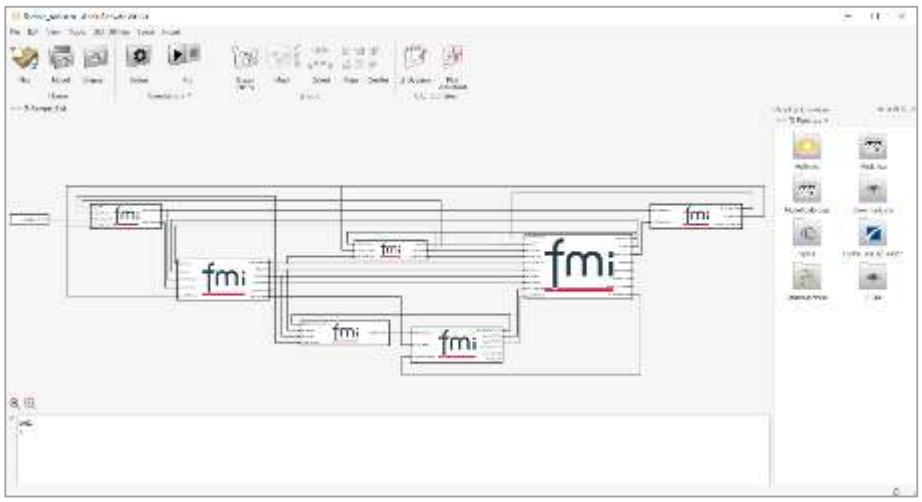


Figure1.4.10 FMU connection model

⑤ Set simulation parameters such as run time and solver via the "Setup" button of "Simulation" menu in the upper ribbon and execute the simulation via the "Run" button of the same menu. If you want to see the simulation results in a graph, connect a "Scope" blockfrom "Signal Viewers" of the Activate library in the Palette Browser to the model.

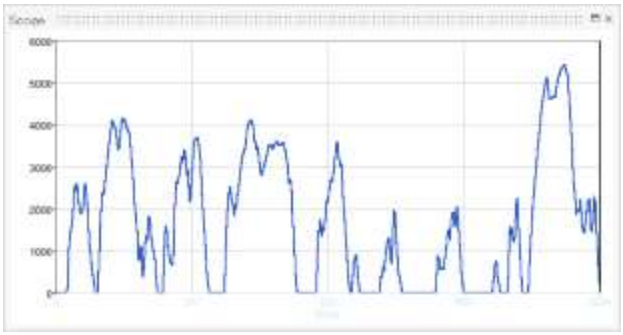


Figure1.4.11 Simulation run results

1.5. dSPACE SystemDesk and VEOS

dSPACE offers SystemDesk as a tool for generating FMUs compliant with FMI 2.0/3.0 and Virtual Ecu Offline Simulator (VEOS) as a tool for simulating FMUs compliant with FMI 2.0/3.0 on a PC, respectively.Table 1.5.1 andTable1.5.2 Table 1.5.1 and Table 1.5.2 respectively.

Table 1.5.1 SystemDesk Interface Features

SystemDesk's FMI creation function	
FMI Version	2.0, 3.0
Form	Co-Simulation
OS	Windows, Linux
Execution License	unnecessary

Table1.5.2 VEOS Interface Features

FMI read function of VEOS	
FMI Version	2.0, 3.0
Form	Co-Simulation
OS	Windows, Linux
Execution License	Depends on the creator

The dSPACE software tool chain for simulating an FMI-compliant FMU is shown inFigure1.5.1 FMI-compliant FMUs generated from SystemDesk and built/simulated by VEOS.

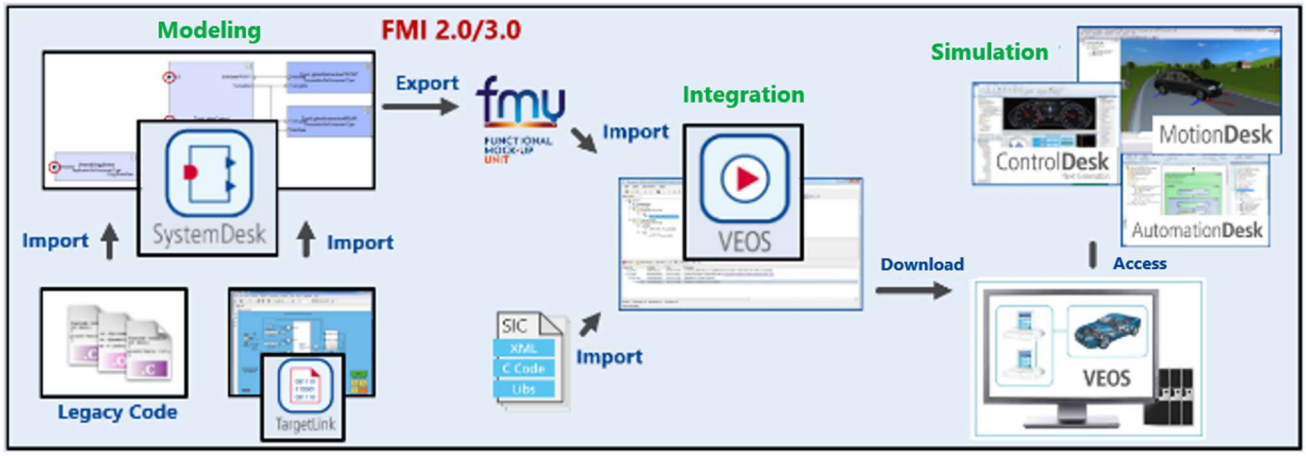


Figure1.5.1 dSPACE's software tool chain

1.5.1. FMU generation with SystemDesk

SystemDesk can generate FMUs compliant with FMI 2.0/3.0.

SystemDesk is a tool with two purposes: first, as an AUTOSAR-compliant software design tool, it allows the design of software components according to Classic or Adaptive AUTOSAR. The second purpose is to generate BSW for simulation and to combine it with the application software so that it

can be used as AUTOSAR compliant software. The second objective is to generate a BSW for simulation and combine it with the application software to see how the application behaves as AUTOSAR compliant software. In this case, SystemDesk generates a container file called Virtual ECU (V-ECU) as a system under test for virtual verification on a PC. In this section, we define an FMI-compliant V-ECU as a "V-ECU FMU".

The following procedure generates an FMI-compliant V-ECU FMU from SystemDesk.
The procedure is shown here using FMI 3.0 as an example.

<Prerequisite>
Necessary files from TargetLink or LegacyCode are already imported into SystemDesk as application software.
<Procedure>
In the ECU Configuration Manager, select ECU Instance and create a new ECU configuration for FMU .

Create a new ECU Configuration for the FMU. Select an appropriate ECU Configuration according to whether or not bus communication is used and the type of bus communication, and then add the necessary BSW modules. The required BSW modules are generated as shown in Figure 1.5.2.

Since the FMI 2.0/3.0 option is available here, it must be selected when generating an FMU.

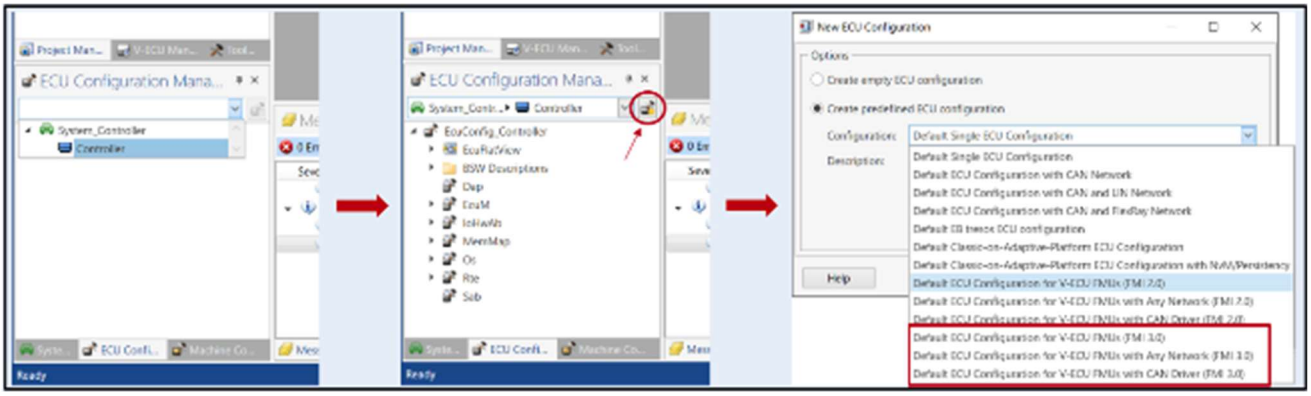


Figure1.5.2 Generating a V-ECU

Edit Runnable Mapping and Generate Mappings for a new ECU configuration(Figure 1.5.3).

This is the first required setting in ECU Configuration.

Each application runnable is associated with an OS periodic task.

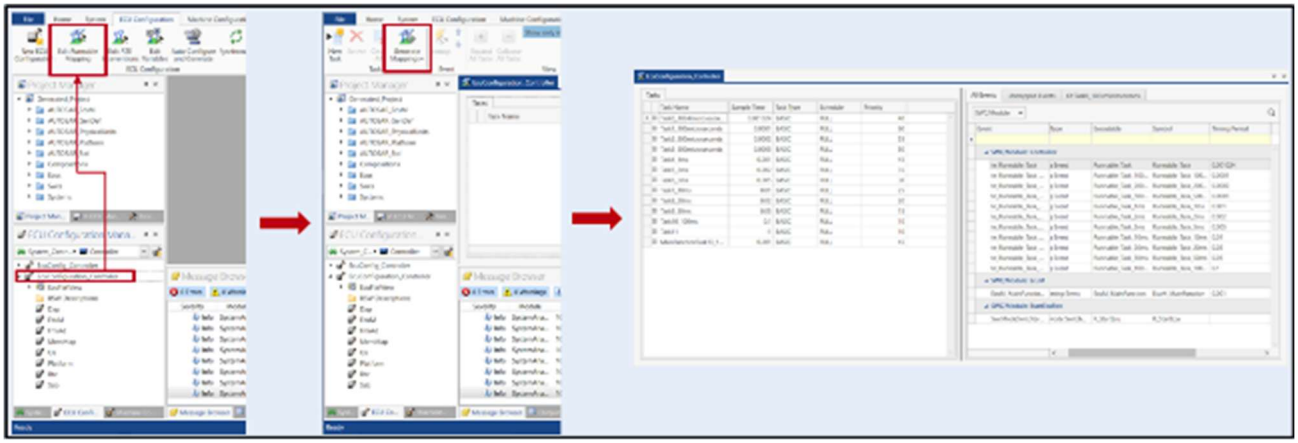


Figure1.5.3 ECU Configuration

・”Edit RTE Interventions”, “Enable Intervention Services”, and “Create RTE Intervention Points"(Figure 1.5.4).

An RTE intervention is a setting that is added to the original RTE code to access and override the communication of a software component when the simulation is run.

For example, when testing a software component, it may be necessary to input a stimulus signal or error generation.

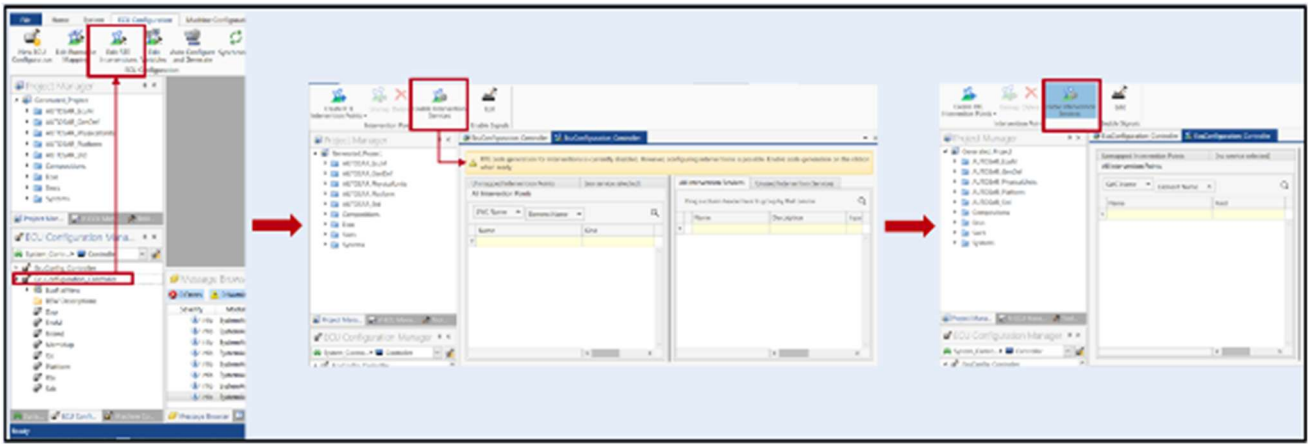


Figure1.5.4 RTE Configuration

Select ...and Services (Data Access Point) from the Create RTE Intervention Points pull-down menu.

Execute as shown in Figure 1.5.5.

Generates I/F signals to be used externally. When built into VEOS (simulation platform), this I/F is used to connect signals to other container files.

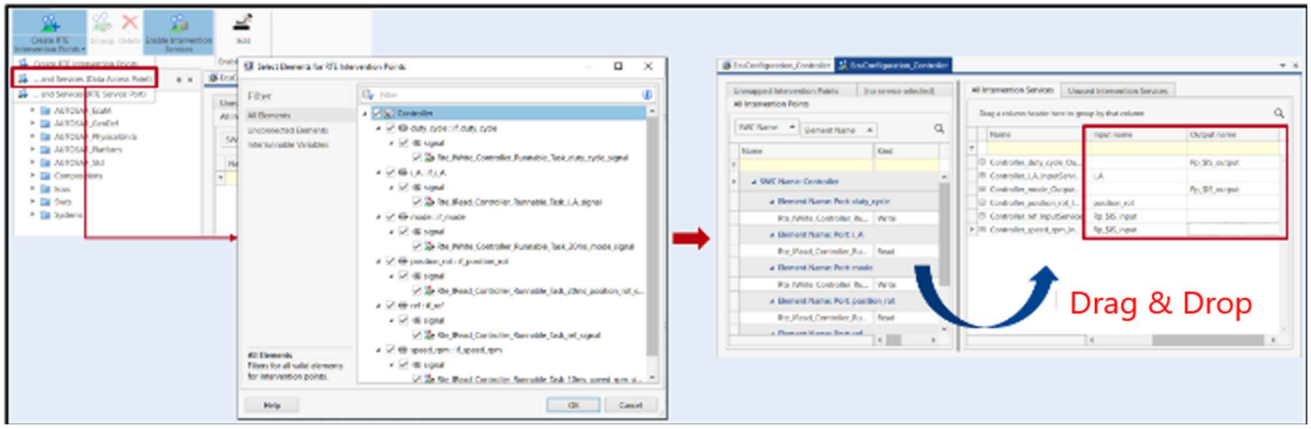


Figure1.5.5 Generation of RTE

・Execute Auto Configure and Generate.

The C code of BSW is generated according to the settings made in ECU Configuration(Figure 1.5.6).

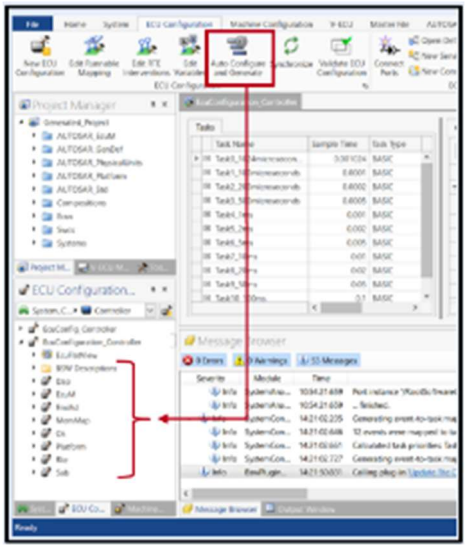


Figure1.5.6 C code generation for BSW

The FMU container from the V-ECU Manager. Create an FMU container from V-ECU Manager as shown in Figure 1.5.7.

Hierarchizes and stores C code, a2l files, etc. required for V-ECU container files.

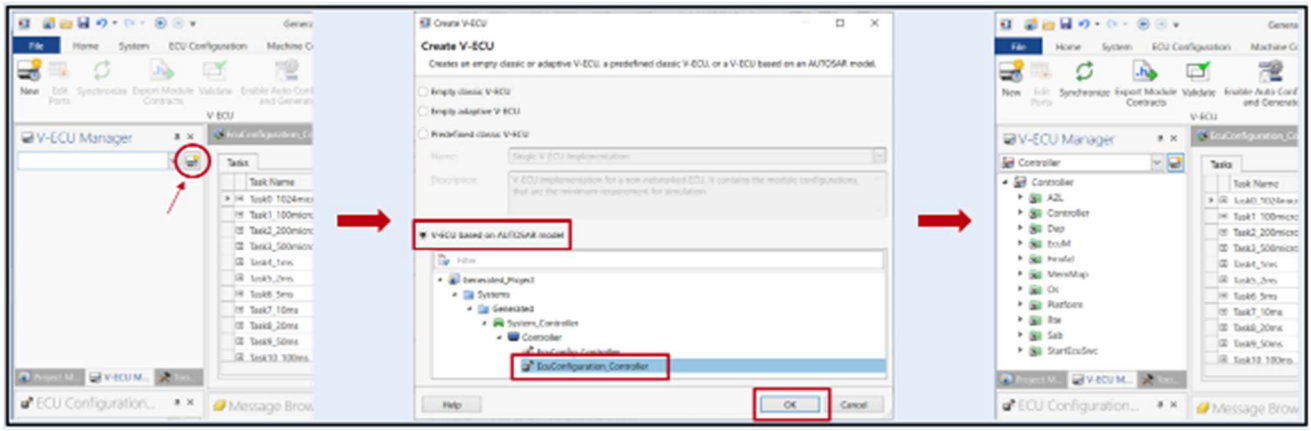


Figure1.5.7 FMU Container Creation

・Finally, the created FMU container is Exported as shown in Figure 1.5.8. (Generation of V-ECU FMU)

Select the platform environment for simulation using the V-ECU FMU, such as Windows/Linux, GCC/MSVC, 32bit/64bit, etc.

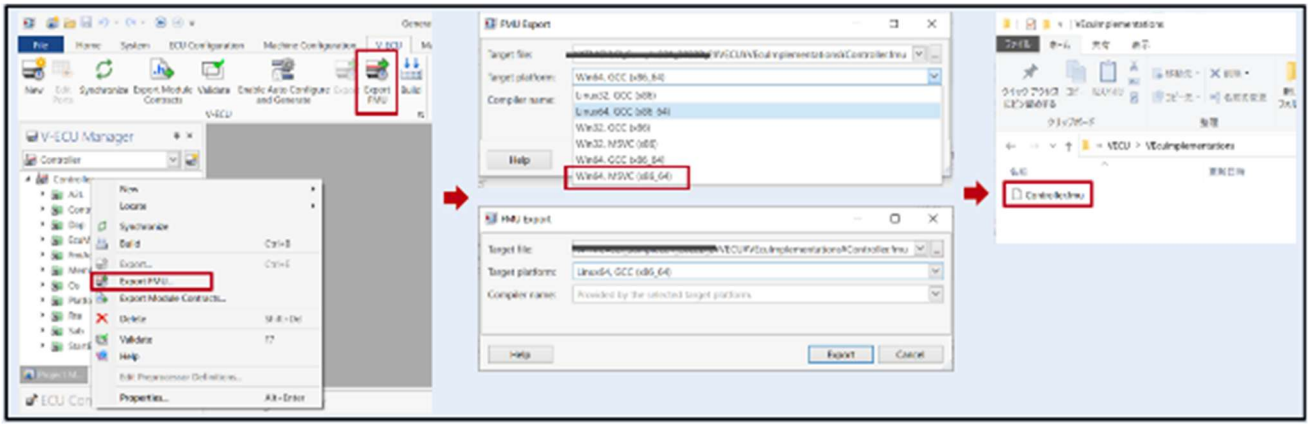


Figure1.5.8 V-ECU FMU generation

1.5.2. FMU Simulation with VEOS

FMUs compliant with FMI 2.0/3.0 can run simulations in VEOS.

VEOS is a PC-based simulation platform that can simulate a wide variety of models from control models to virtual ECUs (V-ECUs), bus systems including networks, and vehicle models, even in the early stages of development without actual vehicles. The system can simulate a wide variety of models, from control models to virtual ECUs (V-ECUs), bus systems including networks and vehicle models.

Follow the steps below to build the FMU in VEOS and run the simulation.

<Prerequisite>

FMI-compliant FMUs are generated using SystemDesk and other generation tools.

<Procedure>

・Select the FMU to be implemented in VEOS from Import as shown in Figure 1.5.9.

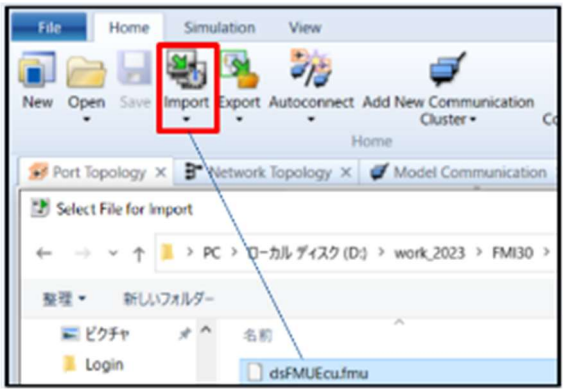


Figure1.5.9 Selecting an FMU

· Select a compiler to be built with VEOS as shown in Figure 1.5.10.

The compiler to be selected should be the same as the compiler used for FMU generation. (Compiler used to create libraries in FMU)

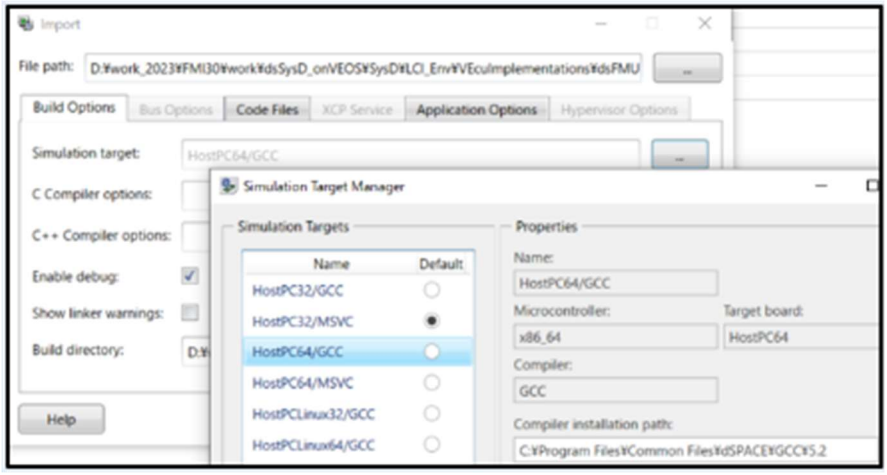


Figure1.5.10 Compiler Selection

・The FMU will be built in VEOS as shown in Figure 1.5.11, and if no errors occur during Build, it will indicate that the FMU has been imported on VEOS.

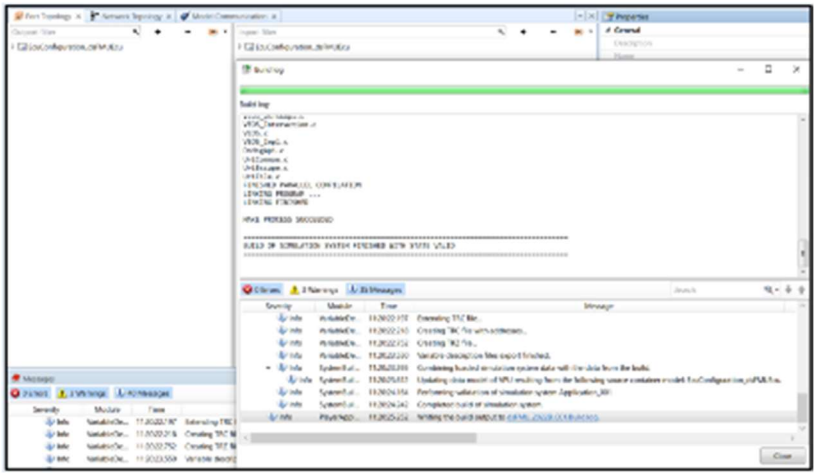


Figure1.5.11 Building the FMU

・By pressing the Start button, The simulation is executed on the VEOS platform as shown in Figure 1.5.12.

By registering VEOS as a platform in a measurement tool such as ControlDesk, you can check the behavior of FMUs running on VEOS.

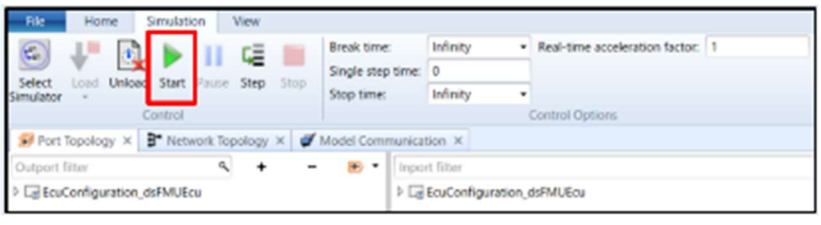


Figure1.5.12 Simulation Start

1.5.3. Simulation using FMI 3.0 with VEOS

When implementing an FMI3.0 compliant FMU in VEOS and running simulations, the following implementation and simulations can effectively utilize the FMI3.0 specification.

1) Use of arrays and integer data types

In FMI 2.0, only Real, Integer, String, and Boolean data types are supported, and in cases where arrays are used, the array elements are broken down into scalars when implemented in VEOS.

For example, if an array with 5 elements is an I/F, it will be broken down into 5 elements after VEOS build(Figure1.5.13). In this case, when connecting I/Fs, connections must be made for the number of signals decomposed.

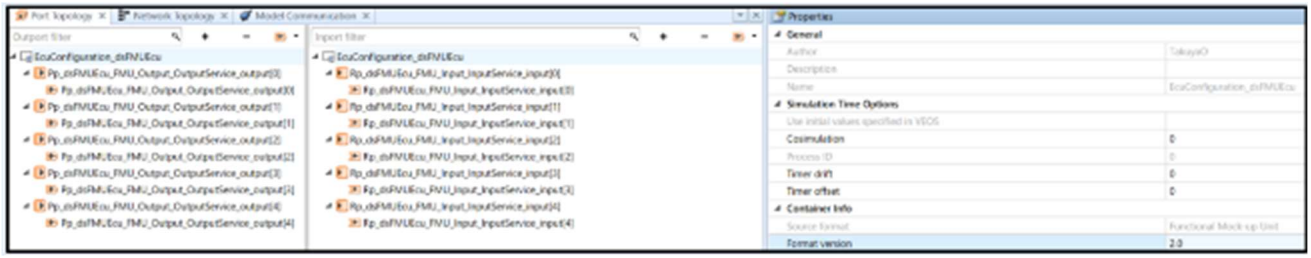


Figure1.5.13 Implementation of an FMI2.0-compliant FMU

On the other hand, if logic with a similar I/F configuration is created as an FMI 3.0-compliant FMU, it is array-compatible, so after VEOS build, the implementation is as shown in Figure 1.5.14.

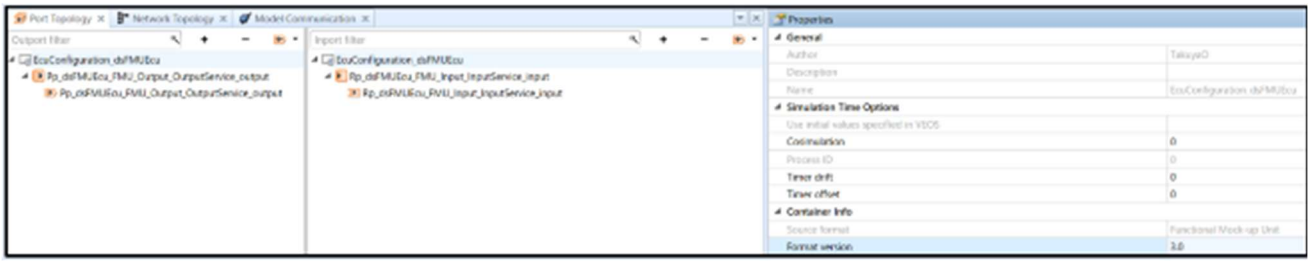


Figure1.5.14 FMU3.0 compliant FMU implementation

2) Using Binary I/F and Clock

FMI3.0 newly supports Binary data and Clock signals, which can be utilized to build a mechanism to exchange sensor data via bus I/F.

An example of an implementation is shown inFigure1.5.15. Two FMUs are connected via the CAN bus, sending and receiving binary data, and triggered by a clock signal.

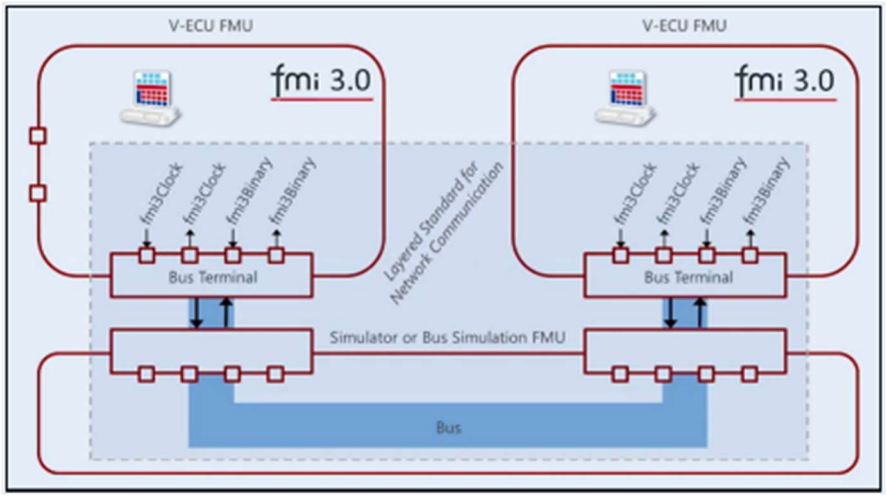


Figure1.5.15 FMI3.0 implementation of Binary and Clock signals

3) Sensor simulation with Binary I/F and OSI connected

FMI3.0 connects to the FMU's Binary I/F, for example, the ASAM standard Open Simulation Interface (OSI), Figure 1.5.16 shows how signals can be sent and received.

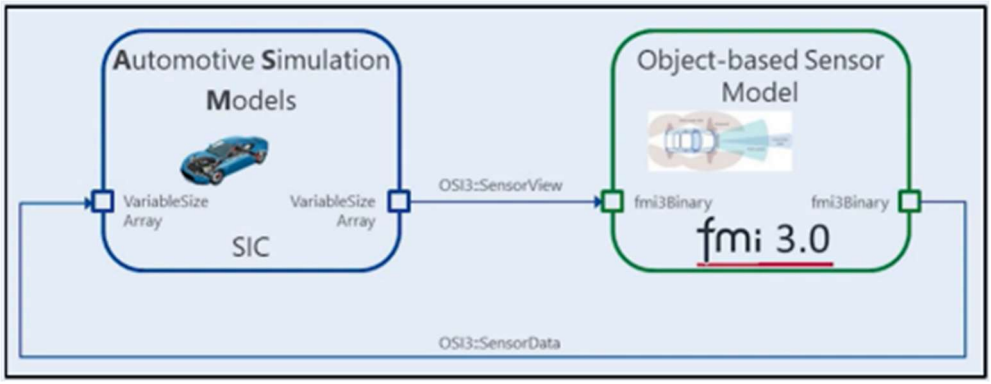


Figure1.5.16 Binary I/F connection using ASAM OSI

Using FMI3.0's Binary I/F, you can realize a system in which SensorView information is sent to and received by a sensor model using OSI from a model that includes the vehicle and associated traffic scenarios, such as dSPACE ASM Traffic, and the sensor model using FMI3.0 detects and feeds back the results. The results of the traffic simulation using OSI are shown inFigure 1.5.17.

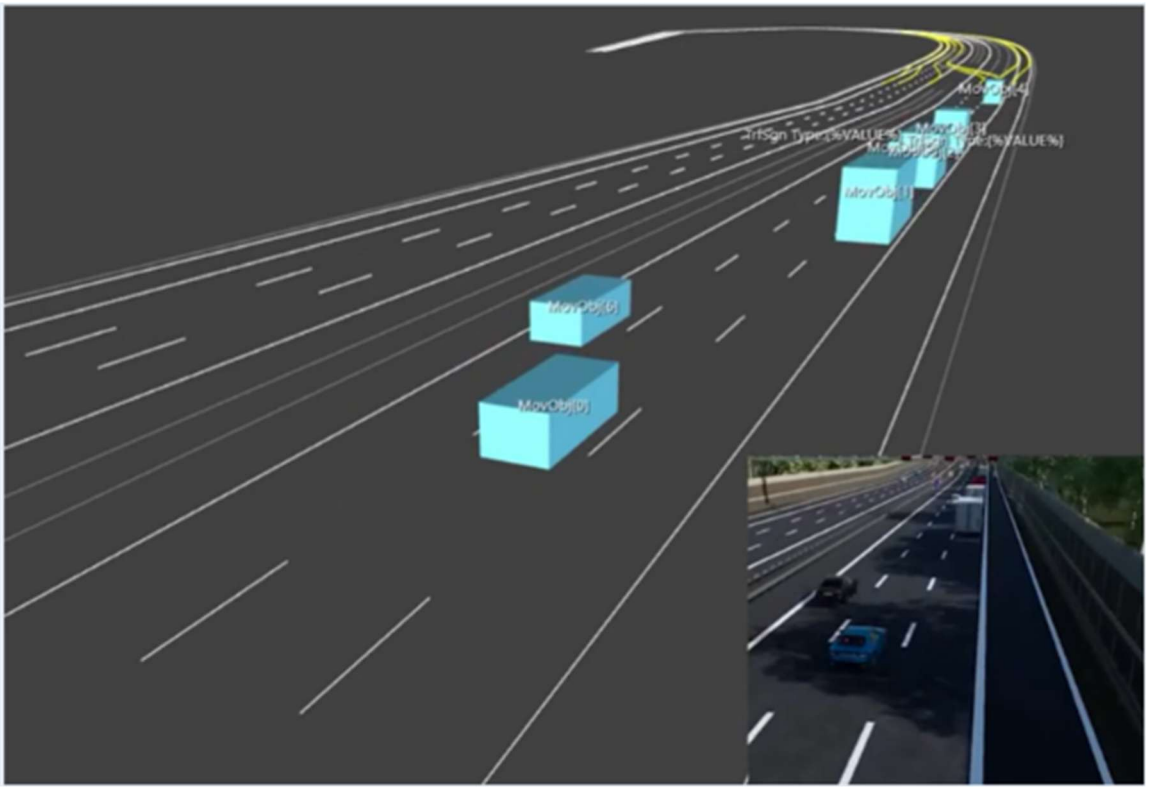


Figure 1.5.17 Traffic simulation results using OSI

1.6. ESI SimulationX

SimulationX is a simulation tool using ESI's Modelica language. The tool is paid for. This presentation is based on version 4.2. The features that this version has, Table 1.6.1 shows the features of this version.

Note that the license structure has changed since SimulationX version 4.4. Please contact the license provider for details.

Table 1.6.1 SimulationX FMI Interface Feature List

FMI Creation Function	
FMI Version	1.0, 2.0
Form	Model Exchange, Co-Simulation
OS	Windows 64, 32
License	At the time of FMU creation: Optional license "Code Export for FMI" is required
	At runtime of created FMU: Not required
FMI read function	
FMI Version	1.0, 2.0
Form	Model Exchange, Co-Simulation
OS	Windows 64
License	When reading FMU: Not required
	At runtime of loaded FMU: Depends on the creator

1.6.1. Creating FMUs

This section describes how to create an FMU. First, activate the optional license "Code Export for FMI". Then, select the menu SIMULATION > Export > Code Export Wizard.

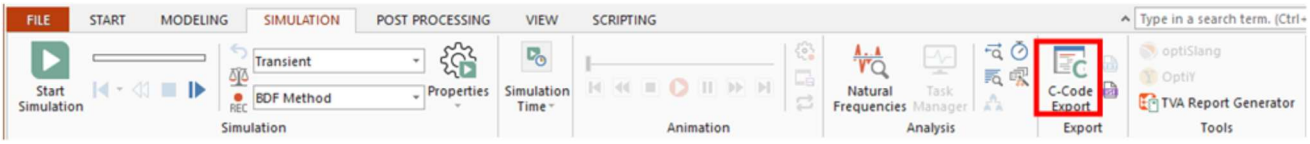


Figure1.6.1 Accessing the Code Export Wizard

In the Code Export Wizard > Project, select "FMI for Model Exchange" or "FMI for Co-simulation. Next, specify the Project Name and Project Path (destination folder).

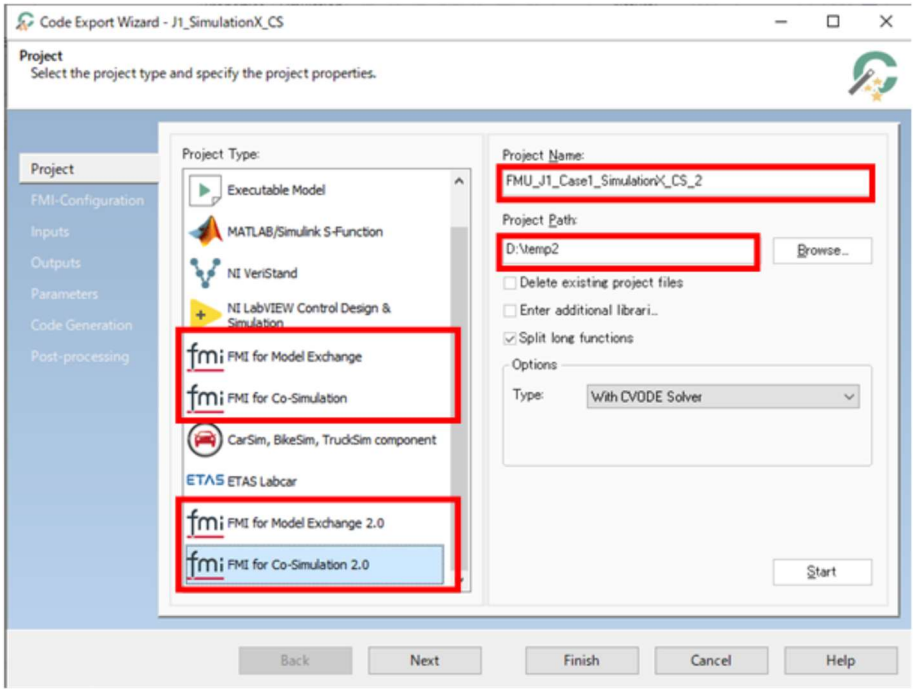


Figure 1.6.2 Project in the Code Export Wizard

In FMI-Configuration, set the creator, version, image, description, etc. of the FMU block; in Platform Support, specify the number of bits in the tool that uses the DLL of the .fmu file you have created.

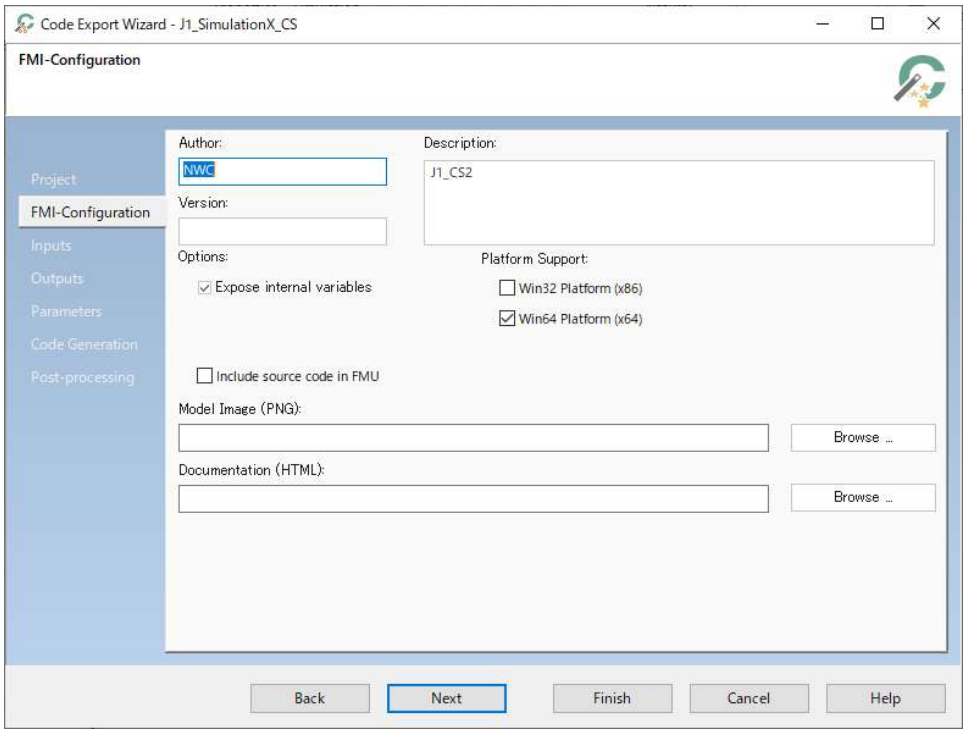


Figure1.6.3 FMI-Configuration in the Code Export Wizard

In Inputs, select the ports to be input into the FMU block. Double-click or Drag & Drop the target port to add it to the "Selection". Only the Signal Input port can be selected.

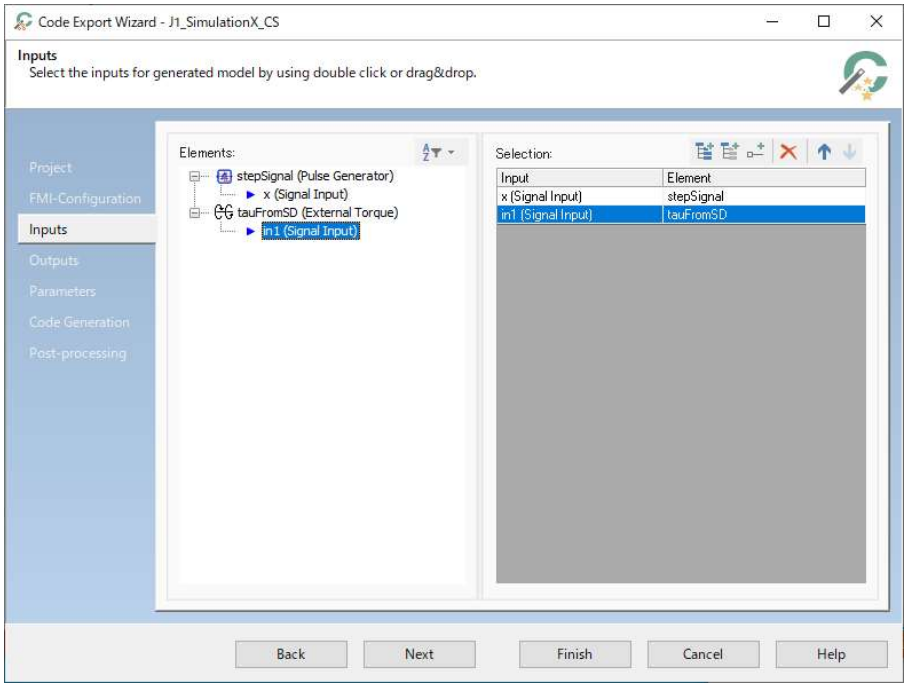


Figure 1.6.4 Inputs in the Code Export Wizard

In Outputs, select the variables to be output from the FMU block. Double-click or Drag & Drop the variable to add it to the "Selection".

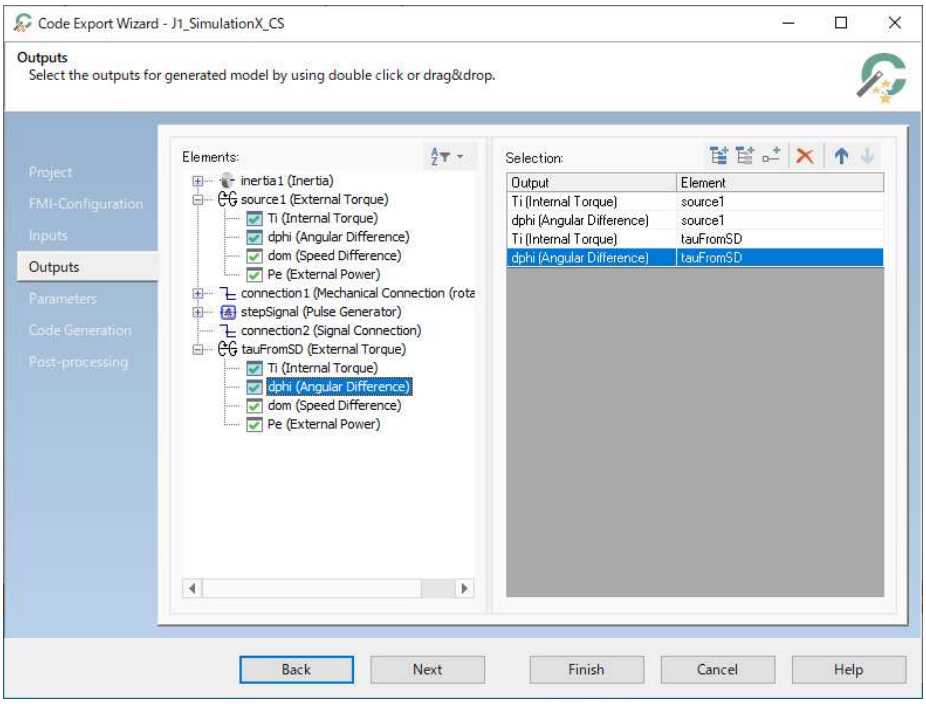


Figure1.6.5 Outputs of the Code Export Wizard

In Parameters, select the parameters of the FMU block. Double-click or Drag & Drop the input variable to add it to the "Selection".

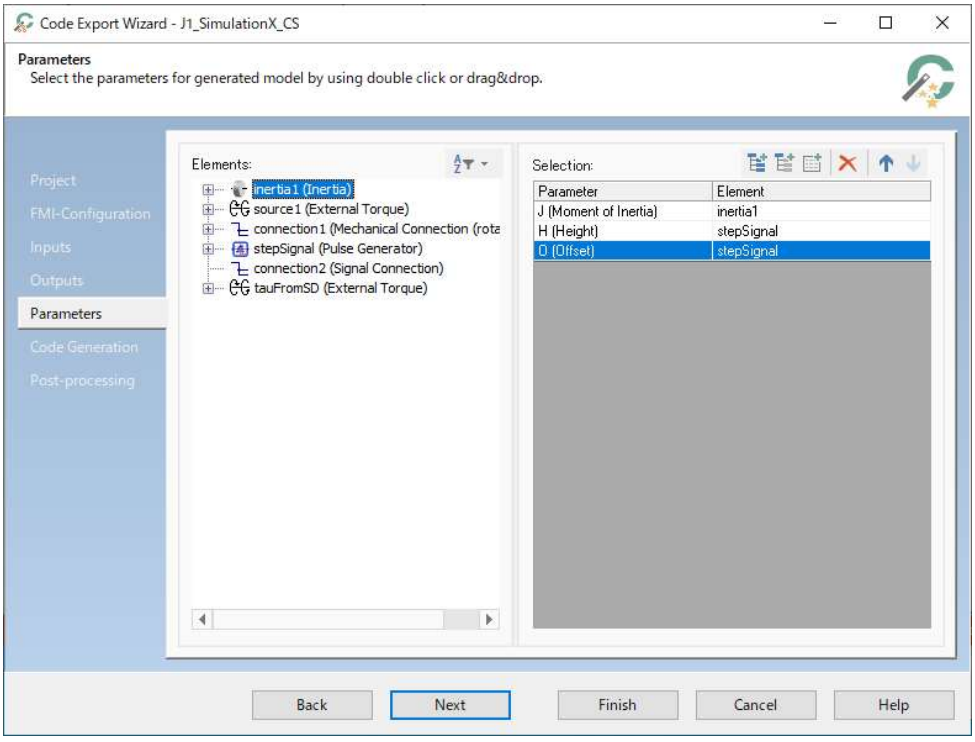


Figure. 1.6 .6 Parameters of the Code Export Wizard

When you move to Code Generation, a sole file of the model is generated. When the generation is completed, the source files and other files are lined up in the folder specified in the Project Path.

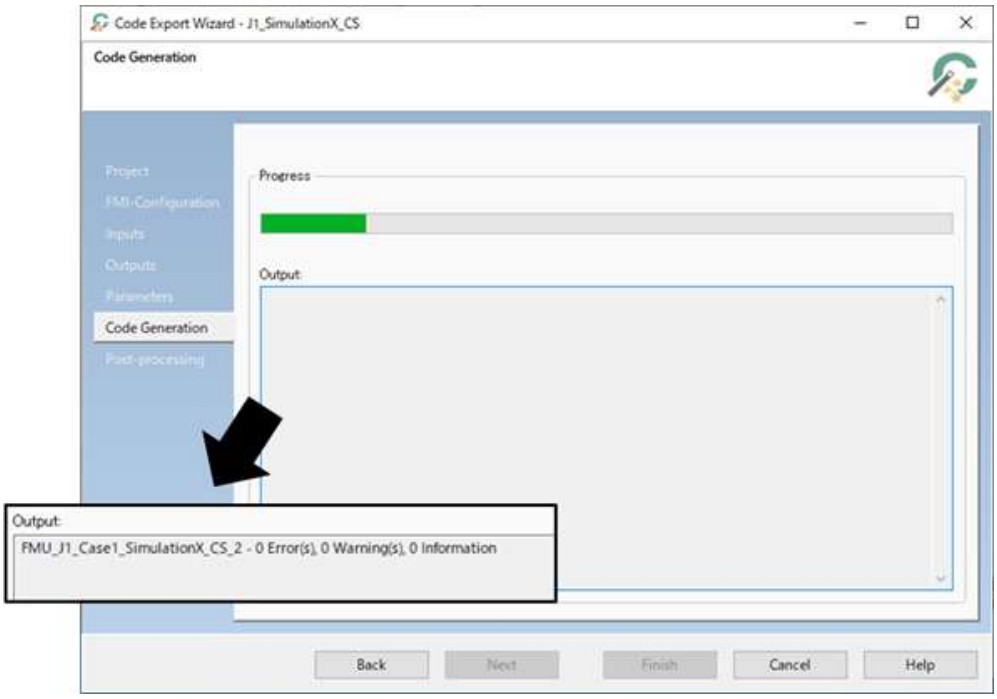


Figure1.6.7 Code Generation in the Code Export Wizard

Compile with Post-processing. After selecting the compiler , press the Build button to start compilation. An FMU is created in the destination folder.

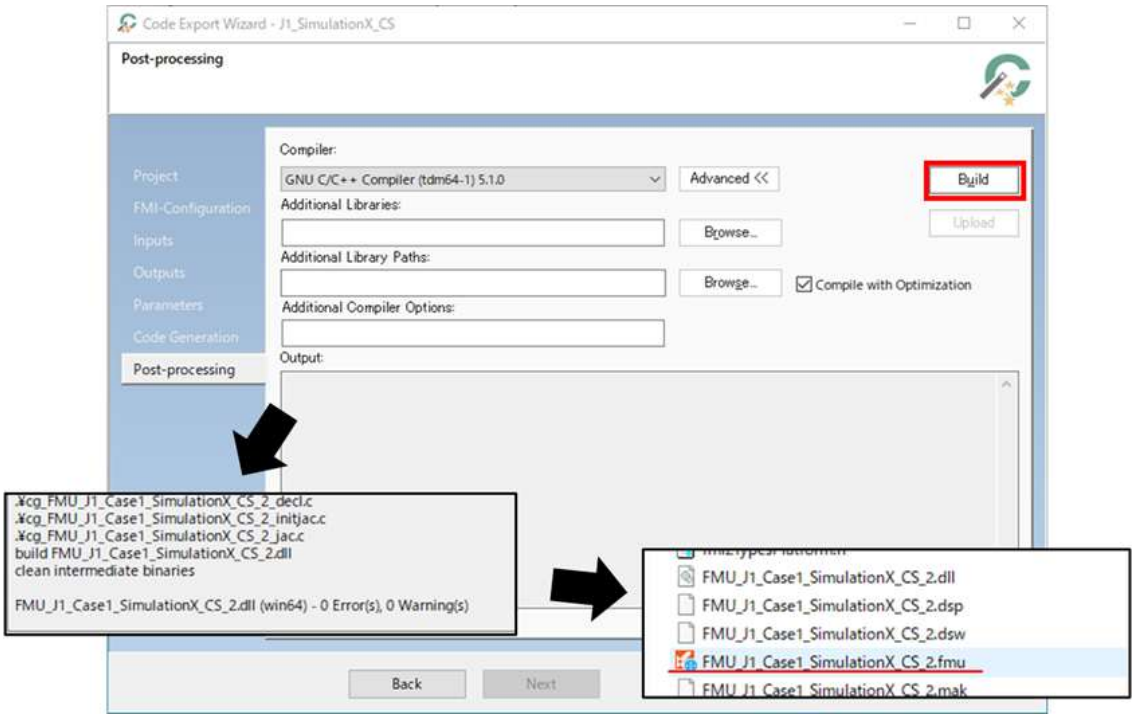


Figure1.6.8 Post-processing in the Code Export Wizard

1.6.2. Importing FMUs

Import functionality is included as a standard feature and does not require an optional license. First, select MODELING > Add FMU Block... Select MODELING > Add FMU Block....

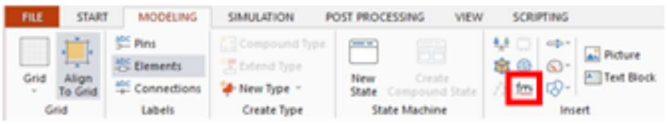


Figure1.6.9 accessing the fmu import function

Next, specify the .fmu file to be imported.

Figure1.6 .10 Specifying the .fmu file to be imported

After specifying the file, select the options to be used when importing.

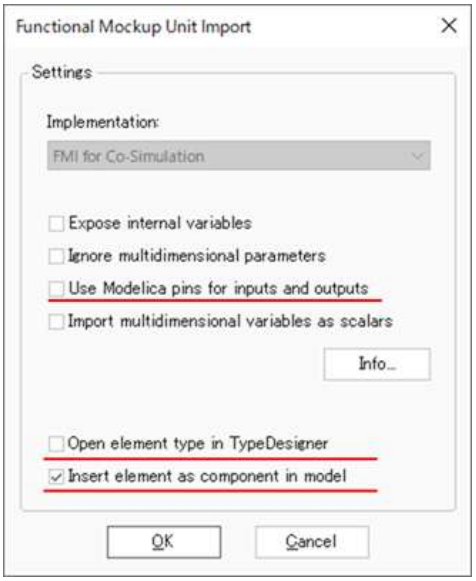


Figure1.6.11 Settings for Functional Mockup Unit Import

Options with/without checks are described below.

Use Modelica Pins for Inputs and Outputs:

Without check, the Signal port of SimulationX is used.

If checked, uses Modelica's Block port.

Open Element Type in TypeDesigner:

Open TypeDesigner and configure port names, parameter names, etc.

Insert Element as Component in Model:

Automatically places imported FMU elements on the Model View.

Repeat the import process for the number of FMUs. Connect the ports of the imported blocks.

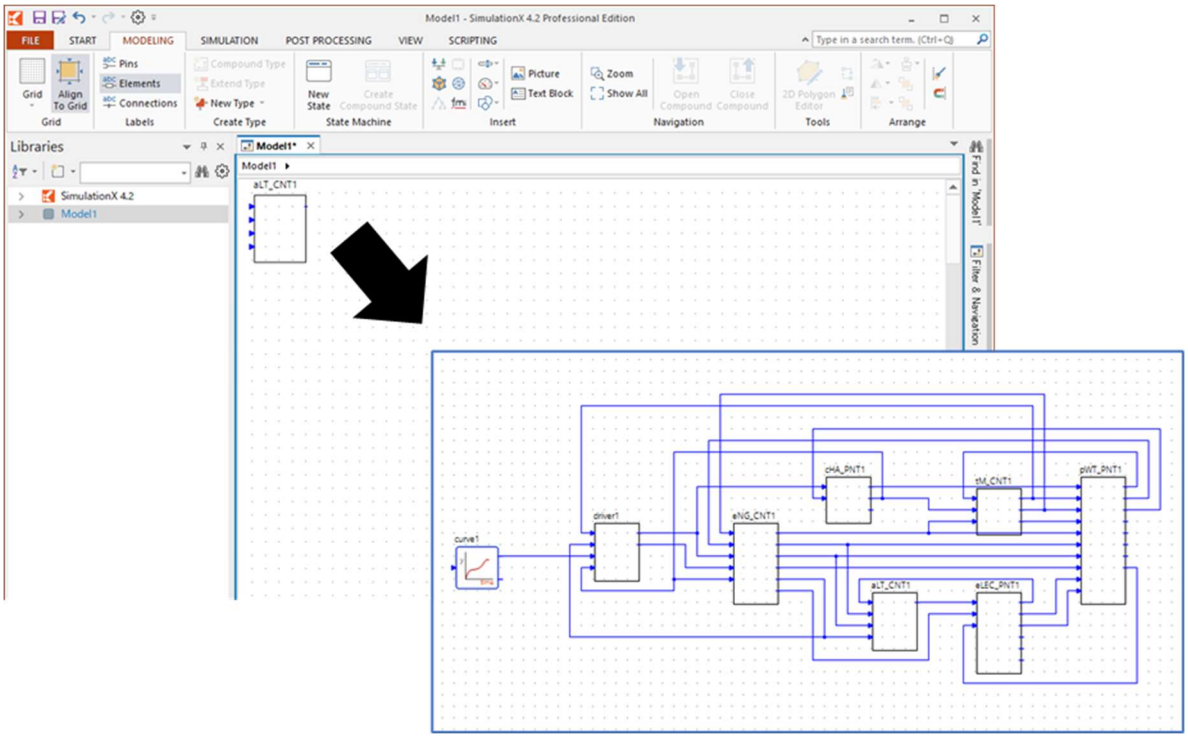


Figure1.6.12 Model Connection Example

1.6.3. FMU Execution

At SIMULATION >Properties, set the simulation conditions.

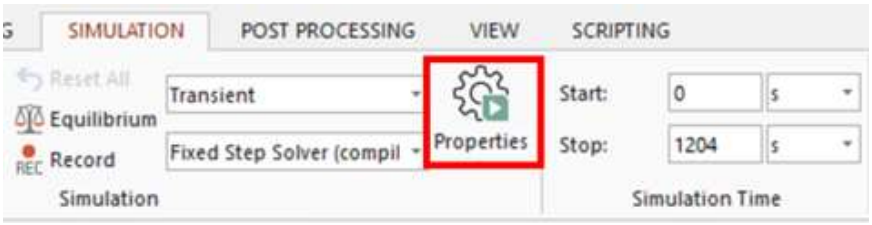


Figure1.6.13 Accessing Properties

In this case, the values to be set are

Start Time : 0 [s].

Stop Time : 1204 [s].

Solver, Step Sizes and Tolerances : Fixed step Solver

Integration method : Euler Forward (Equivalent to ODE1)

Min. Calculation Step Size : 0.0025 [s]

Min. Output Step Size : 0.01 [s] ・・・Arbitrary since it is the sample time of the output result.

Co-Simulation requires setting the time interval for each FMU to receive the input signal (Communication Step Size). Double-click on the model and go to Properties> Master Algorithm. This time, set Communication Step to 0.0025 [s] for all models.

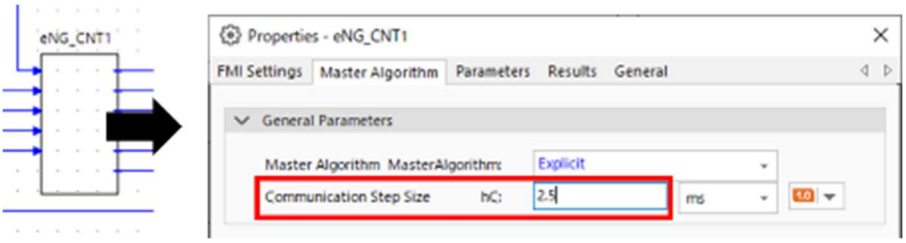


Figure 1.6.14 Properties > Master Algorithm> Communication Step Size

Settings are required to observe the output results during simulation. Double-click on the model and go to Properties> Results. Check the box to enable observation.

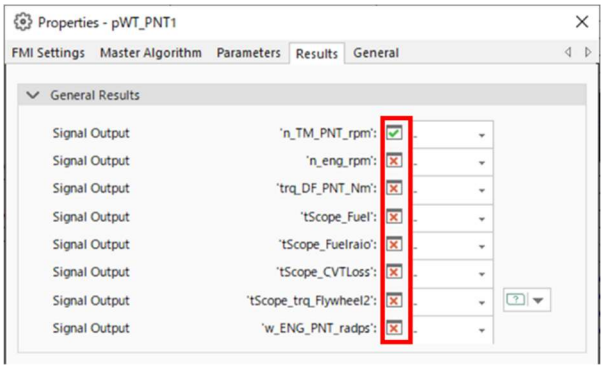


Figure1.6.15 Properties > Results

Run the simulation at SIMULATION >Start Simulation.

Figure1.6.16 Start Simulation

Right-click on each imported FMU, and the output results will be displayed in the Results section.

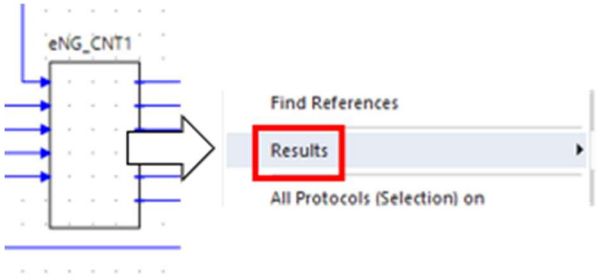


Figure1.6.17 Output Results

1.7 Gamma Technologies GT-SUITE

GT-SUITE is a multiphysics system simulation tool developed by Gamma Technologies, Inc. It can be used for NV evaluation of energy efficiency, noise, etc., and thermal management evaluation of cooling systems under arbitrary operating conditions.

FMI 1.0 and 2.0 are supported. (3.0 will be supported in the future.)

Co-Simulation and Model Exchange are supported for import into GT-SUITE, while Co-Simulation is supported for export from GT-SUITE (FMU generation).

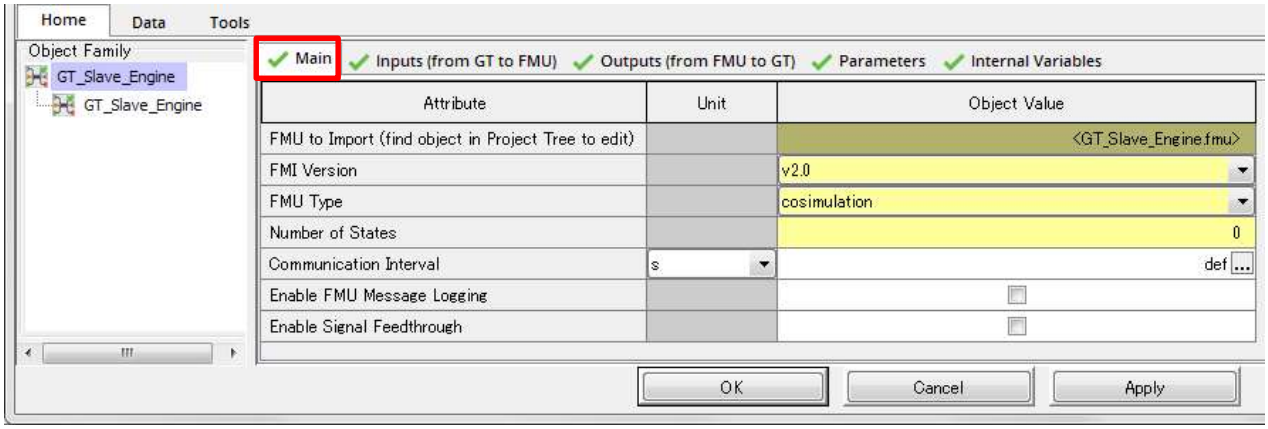
1.7.1 Import to GT-SUITE

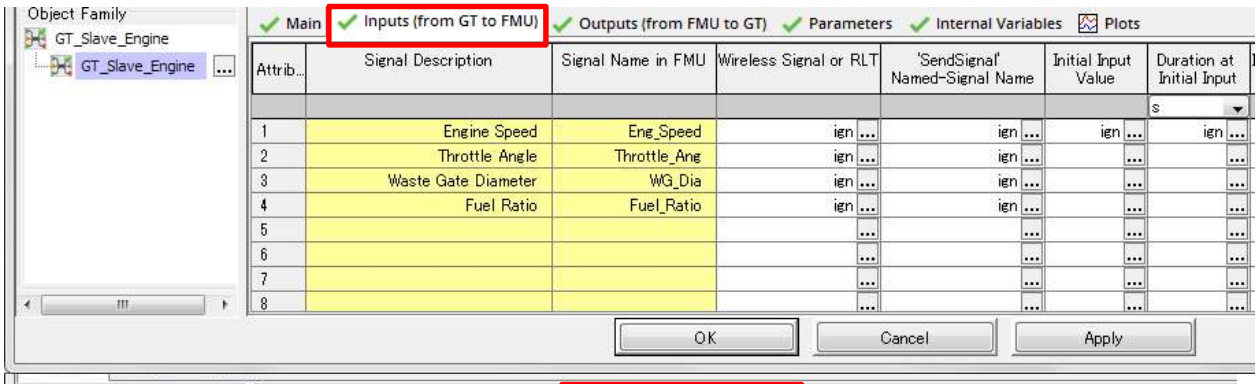
・FMUImport settings

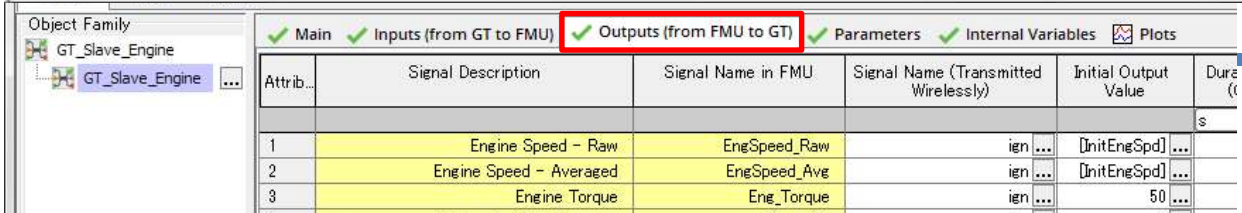
Use the FMUImport object to load FMUs generated by other tools.

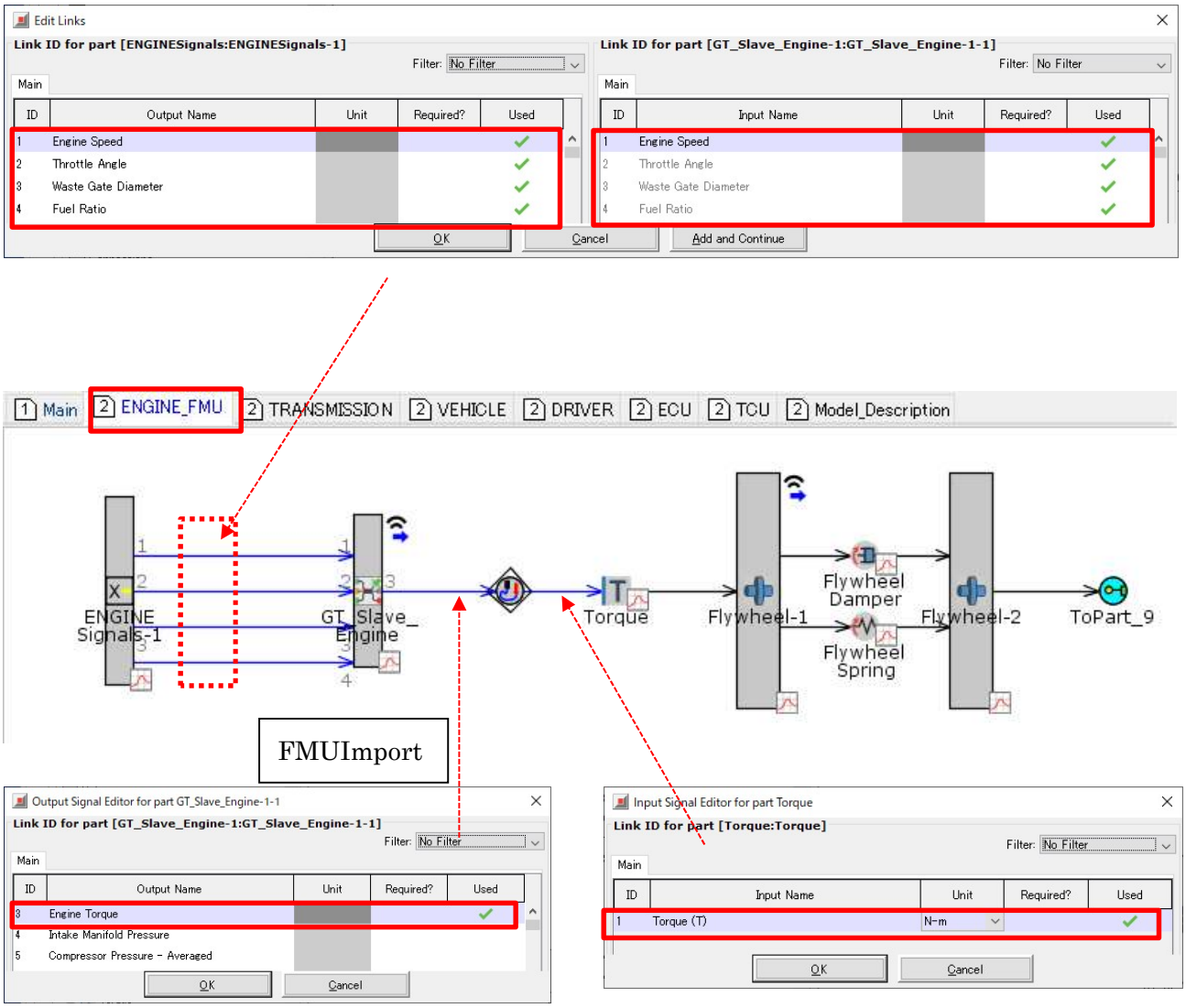
When you select the FMUImport object from the GT-ISE library, you will initially select the FMU (*.fmu file). Then the signal names, etc. will be loaded in the Inputs/Outputs folder.

Next, place the FMUImport as a part. To pass signals to the imported FMU, connect any signal to the FMUImport via SensorConn. (Alternatively, you can use the Inputs folder to receive signals from SendSignal or RLT values.) To receive signals from FMU, connect FMUImport to any part via ActuatorConn. (Alternatively, signals can be sent to individual parts via the Outputs folder.) Multiple input and output signals can be handled.





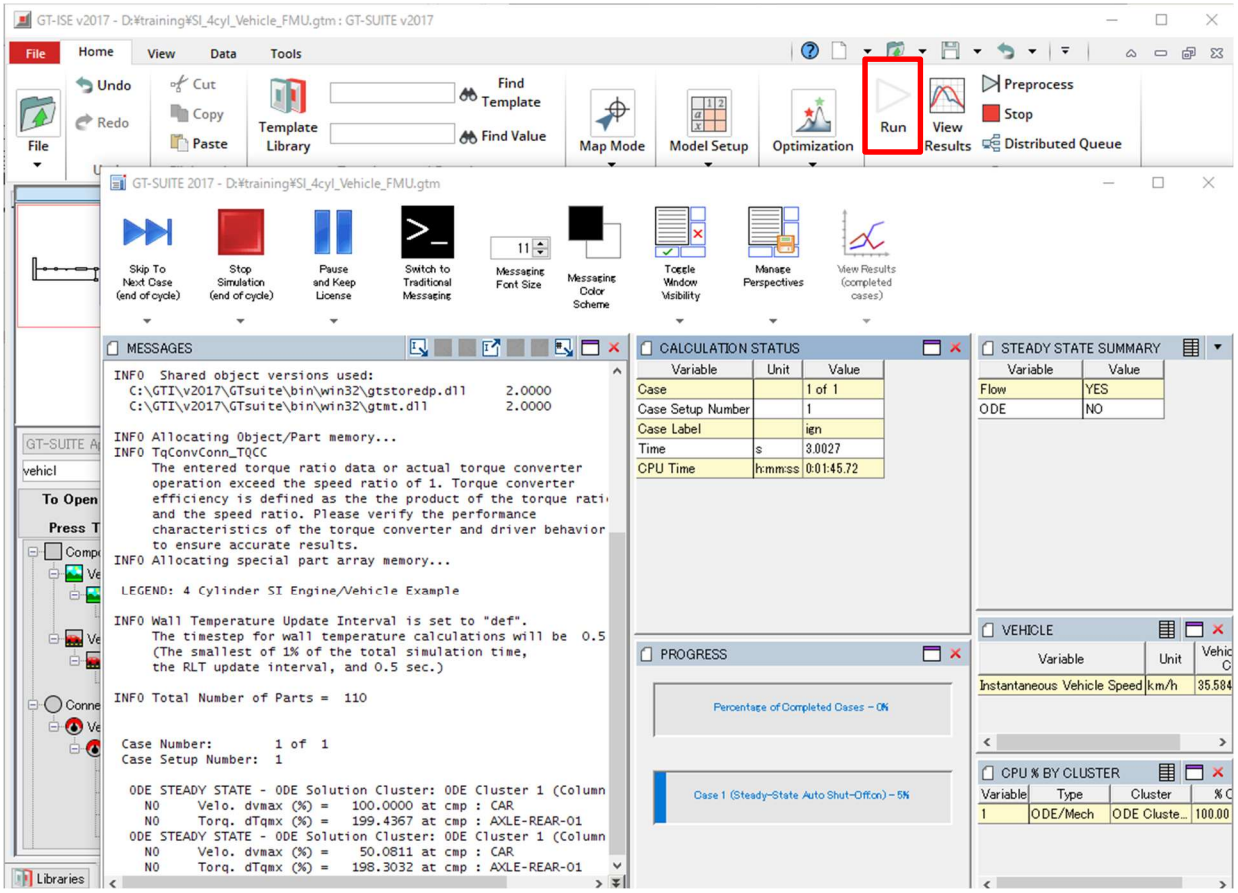




(e.g.) When passing engine speed, etc. to FMUImport and receiving torque

・Calculation execution

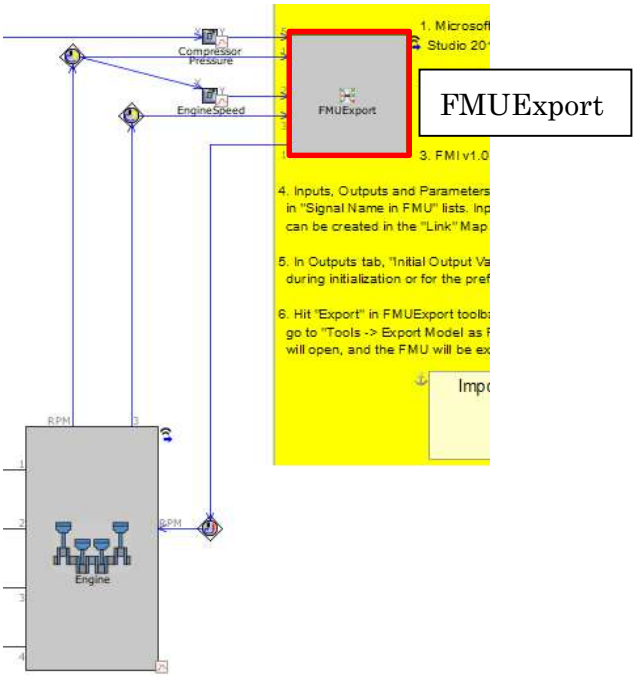
Running the simulation as usual from GT-ISE will start the calculation.



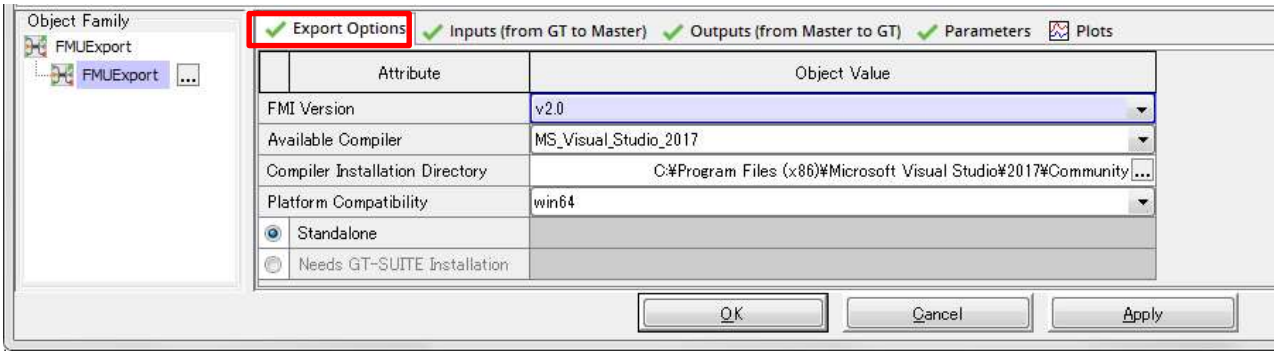
1.7.2 Export from GT-SUITE (FMU generation)

FMUExport settings

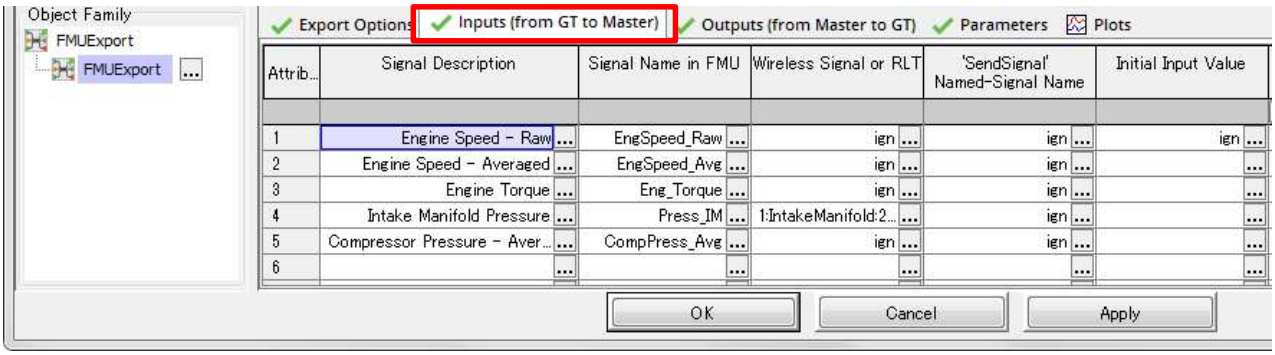
Use the FMUExport object to set up signal passing with other tools.

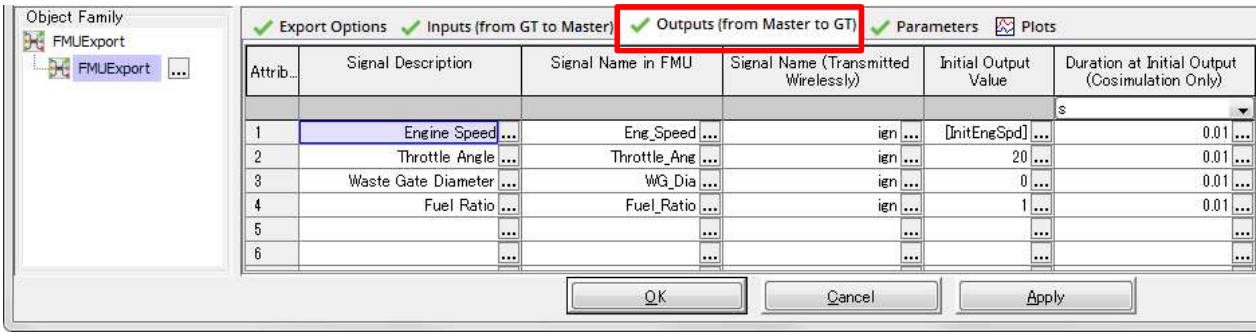


In the Export Option folder, select the compiler to be used in Available Compiler and specify the installation folder in the Complier Installation Directory. Specify the environment in which FMU will be used in Platform Compatibility. The radio buttons allow you to select whether or not the installation of GT-SUITE is required in the FMU user environment.

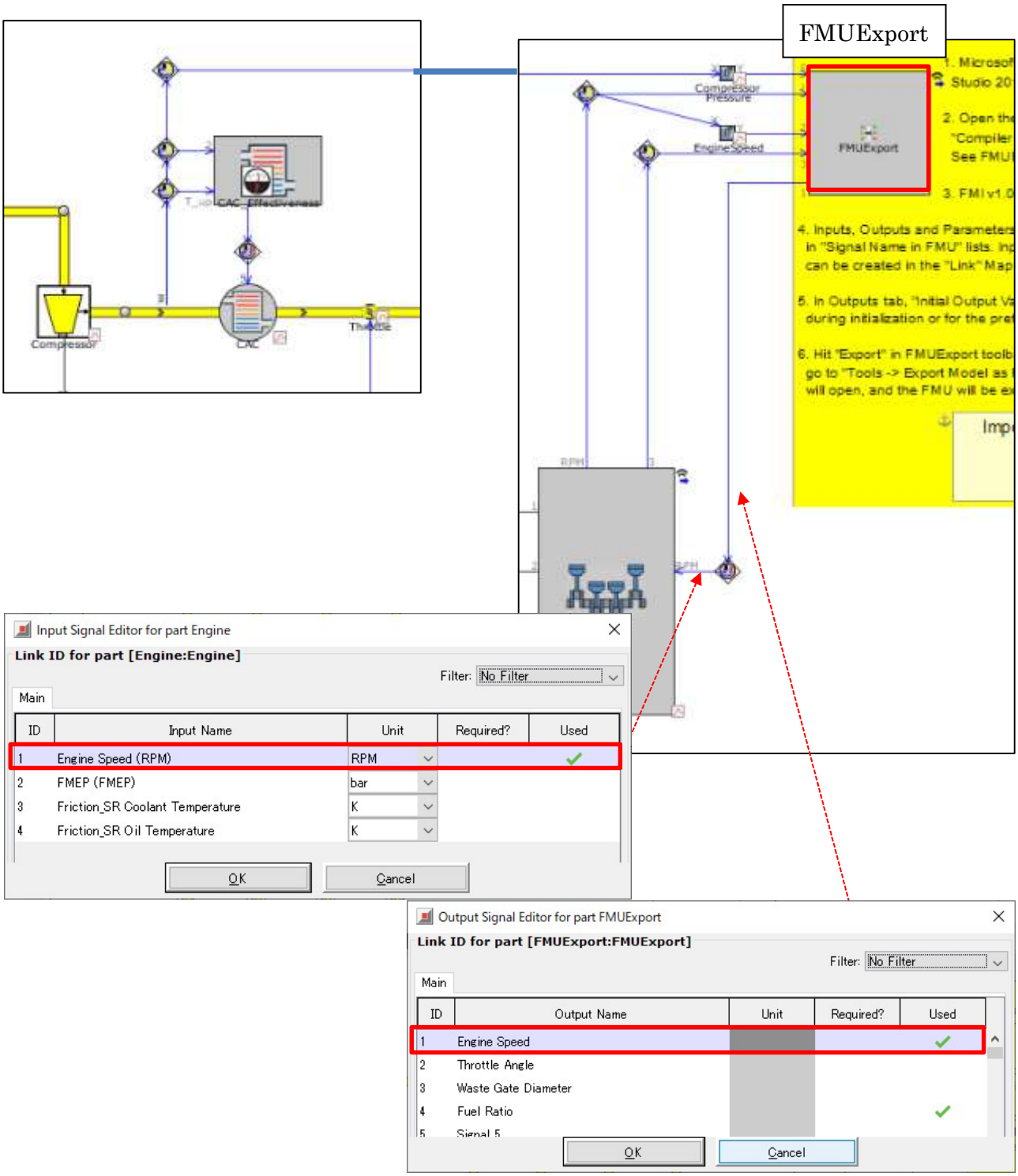


The Inputs/Outputs folder specifies the signals to be passed from GT-SUITE to other tools, and the Outputs folder specifies the signals to be passed from other tools to GT-SUITE. Signal Name in FMU is the signal name of the FMU, which is read by other tools.

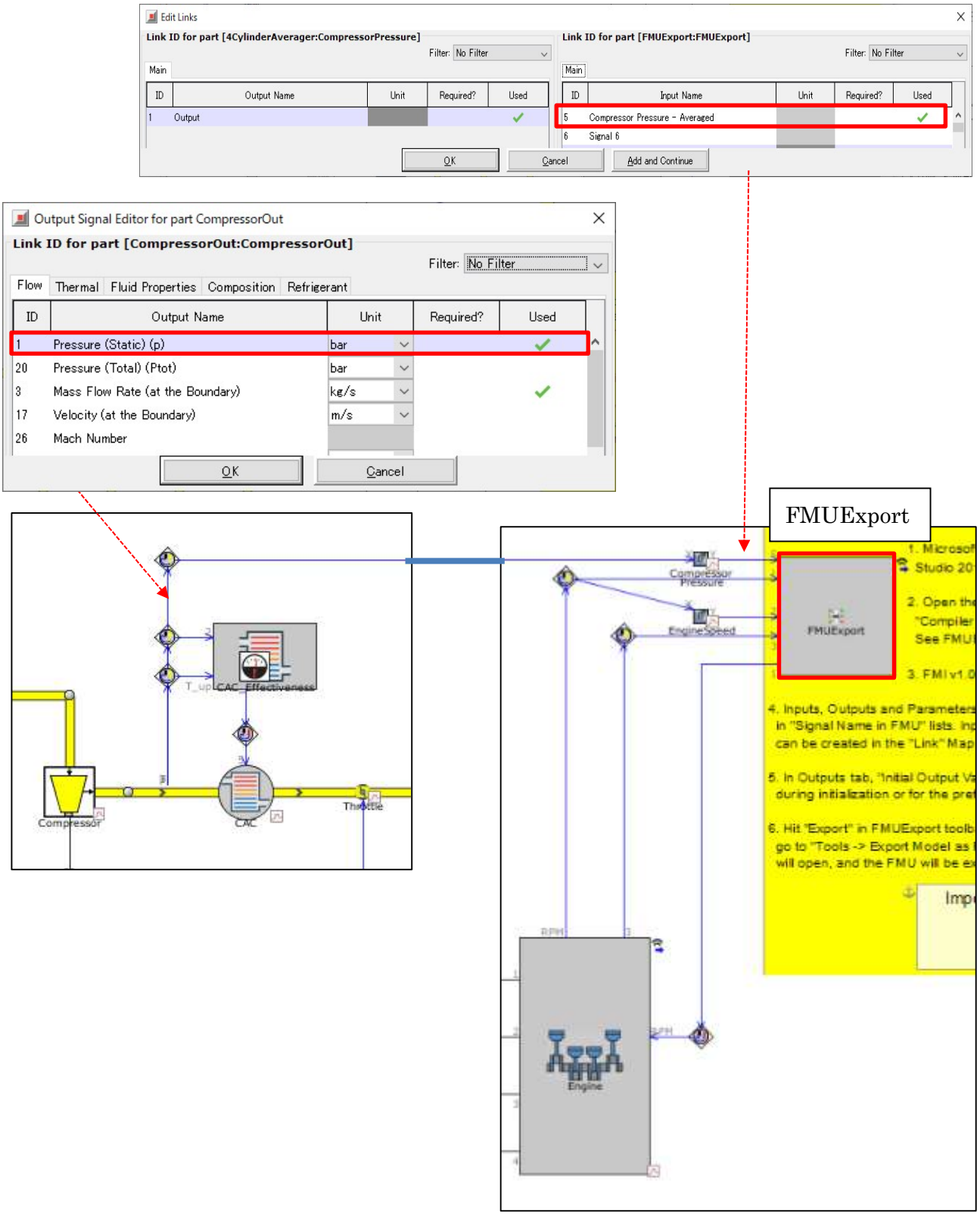




To pass signals to other tools, connect any signal to FMUExport via SensorConn. (Alternatively, you can use the Inputs folder to receive signals from SendSignal or receive RLT values). To receive signals from other tools, connect FMUExport to any part via ActuatorConn. (Alternatively, signals can be sent to individual parts via the Outputs folder.) Multiple input and output signals can be handled.



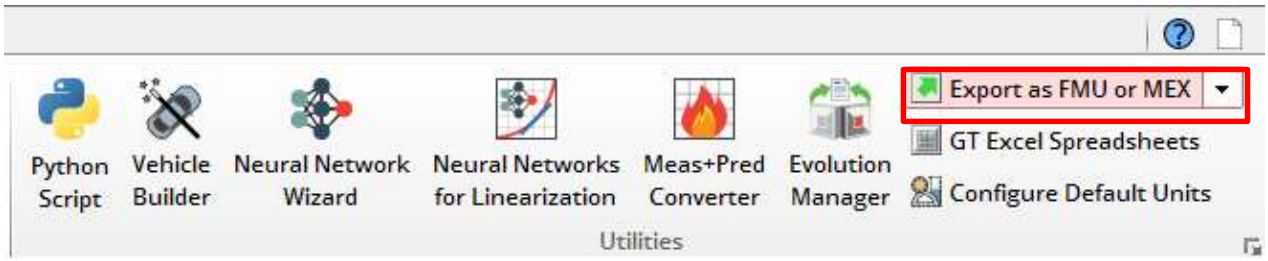
(e.g.) When receiving engine RPM from another



(e.g.) When passing compressor pressure to another tool

・Export

Selecting Export as FMU or MEX from the Tools ribbon in the GT-ISE menu will automatically generate an FMU (*.fmu file).



Generated FMUs can be loaded and used in other tools.

1.8 ETAS VECU-BUILDER

The FMU generation tool VECU-BUILDER provided by ETAS has the following features

- FMI 2.0 compliant
- XCP slave functionality is added to the generated FMU (stand-alone, can be linked with INCA, etc.)
- Generated FMUs can be co-simulated with built-in Solver
- FMU generation from C source code, object files, and DLLs
- Command line execution
- Compatible with Windows and Linux

By specializing in FMU generation, the tool chain can be flexibly configured with ETAS COSYM and other FMI-compliant cosimulators.

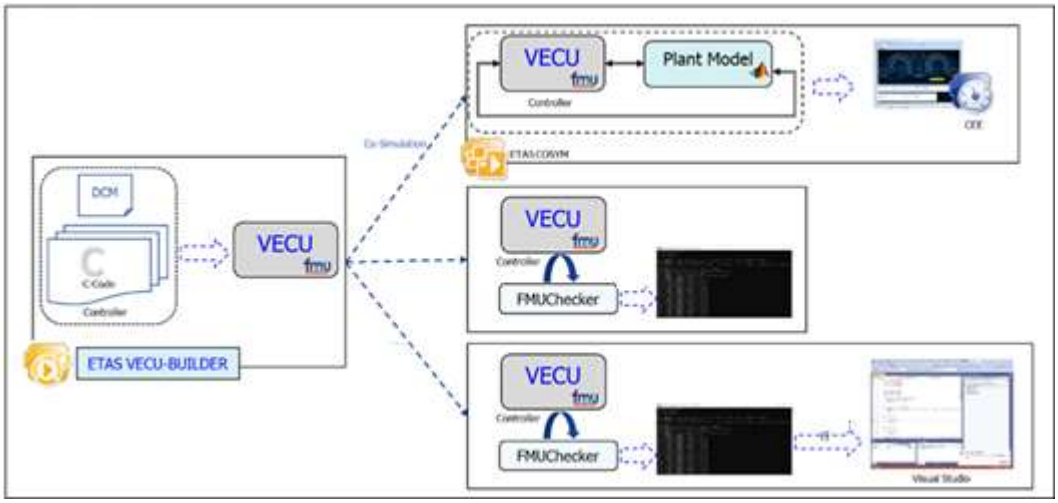


Figure 1.8.1 Example of VECU-BUILDER and tool chain configuration

The command line interface allows for easy integration into the build pipeline for batch processing.

1.8.1 Creating FMUs

VECU-BUILDER generates FMUs from the original FMU files (C source code, object files, DLLs, etc.) and YAML format configuration files.

The YAML file allows detailed configuration of input files, input/output and referenced signals, task cycles, EEPROM operation, external include files, etc.

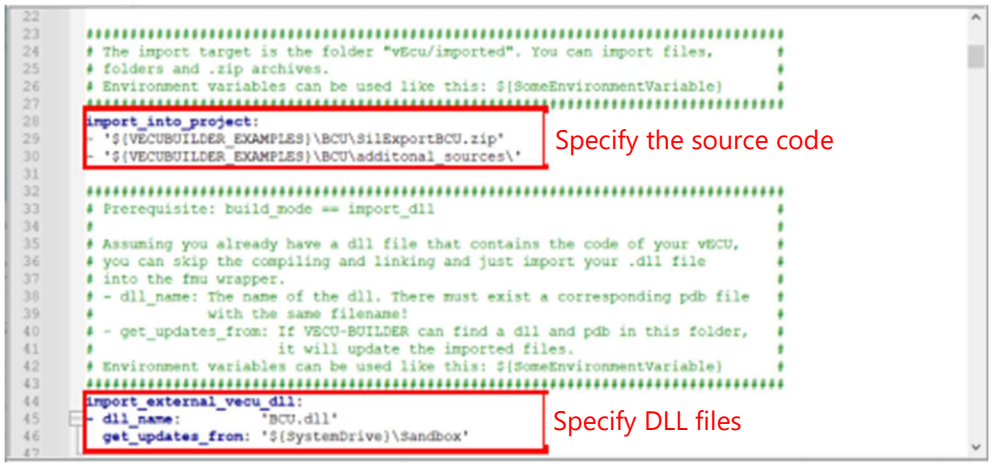


Figure1.8.2 Example YAML File: Input File Configuration

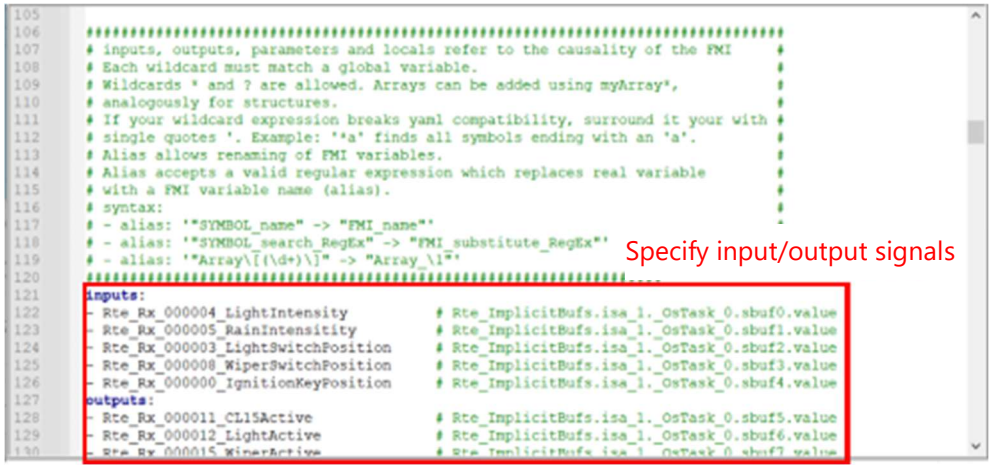


Figure 1.8.3 YAML File Example: Input/Output Signal Configuration

FMUs are generated by command line execution, and a series of necessary operations such as building FMUs, simple operation checks, and debugging are provided in VECU-BUILDER as batch files.

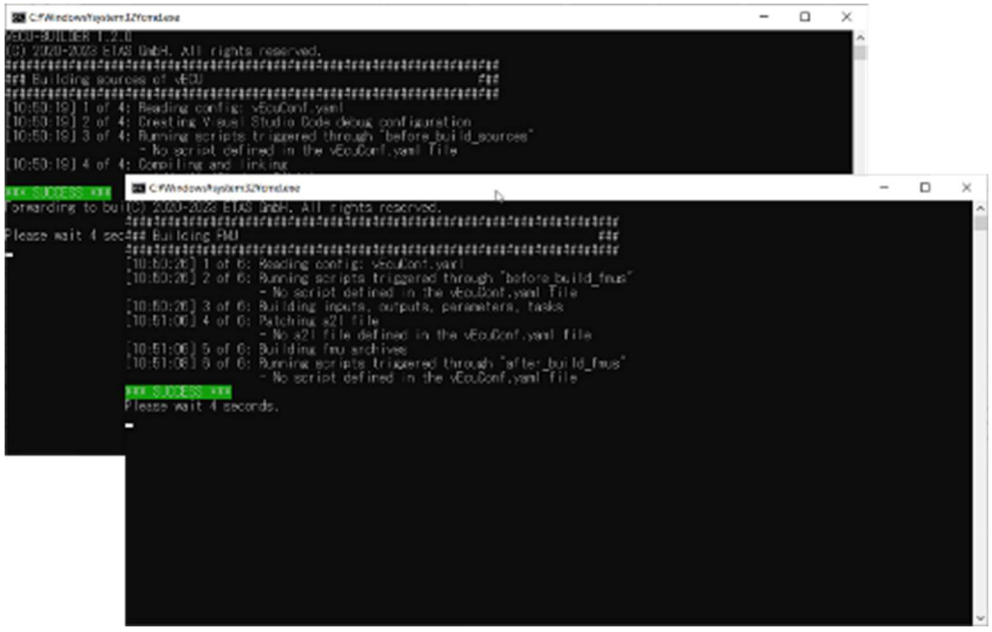


Figure1.8.4 FMU build (command line execution)

1.8.2 FMU import and execution

An FMI-compliant co-simulator is used to run the FMU.

The ETAS cosimulator COSYM is used here.

First, import the FMU you wish to run. This can be done from the "Import model" context menu.

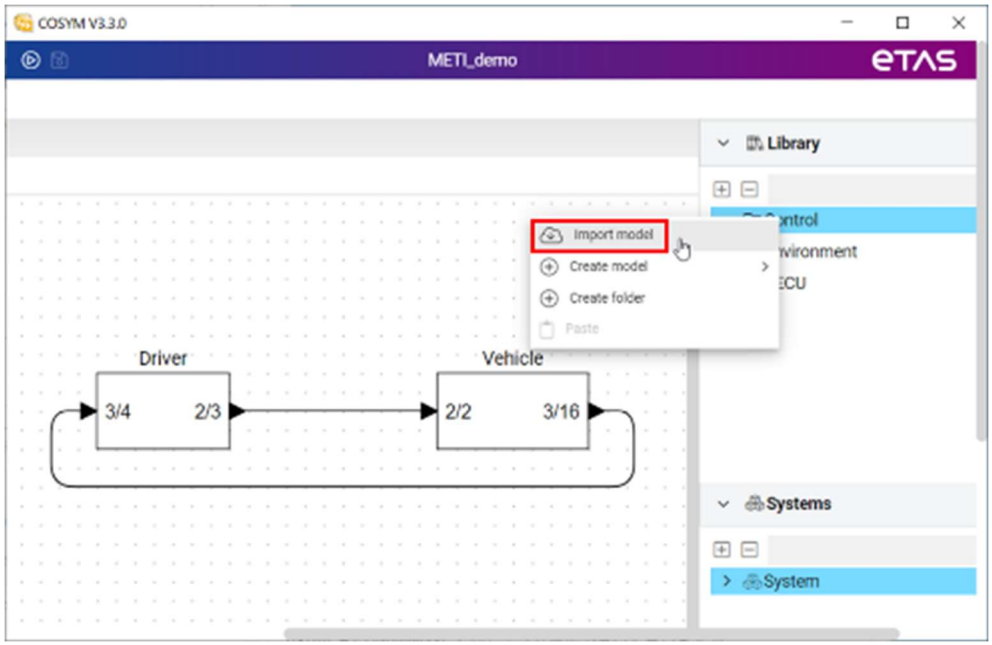


Figure1.8.5 Importing FMUs

Next, the imported FMUs are added to the system to be simulated. This can be done by drag-and-drop from the list of imported models to the design screen.

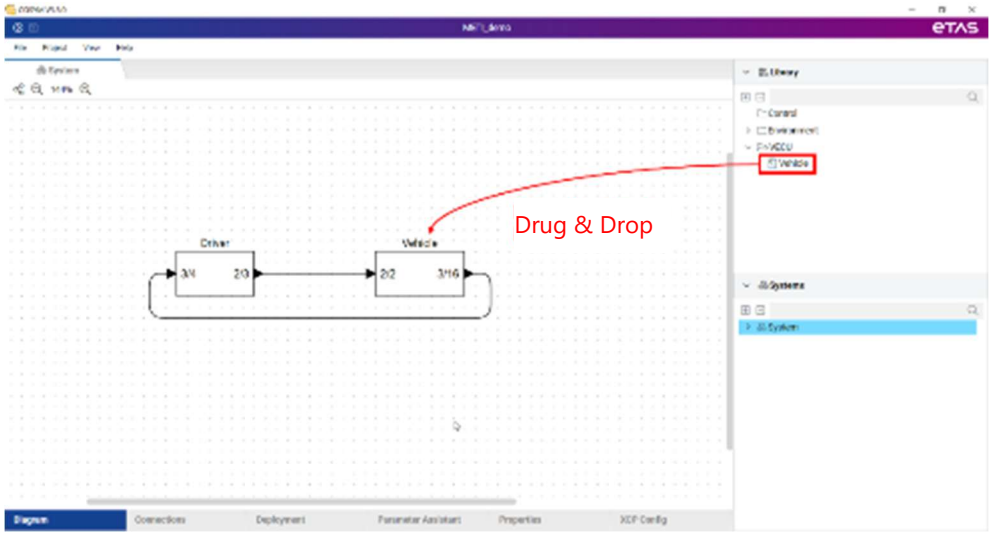


Figure1.8.6 Placement of FMUs in the system to be simulated.

It then sets the signal connections and the order in which tasks are executed.

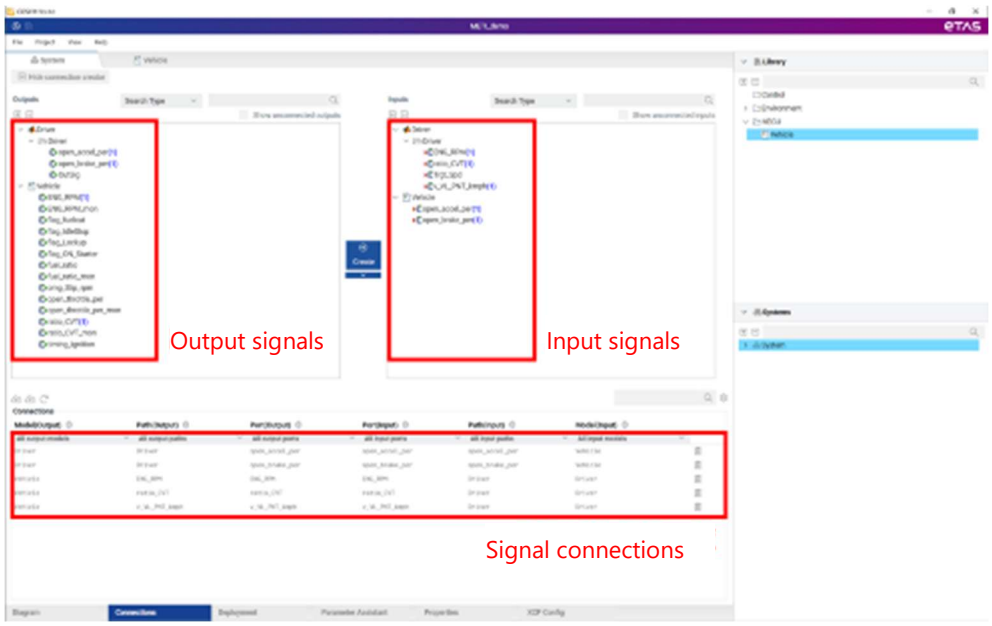


Figure1.8.7 Signal connections between blocks

When all of the above is completed, click "Build and Run. The simulation experiment screen will be launched, allowing you to verify the operation of the simulation system designed in the previous screen. The GUI configuration, such as input signal manipulation and display of monitored signals, is also performed on the experiment screen.

Figure1.8.8 Simulation Run Screen

1.9. Modelon Modelon Impact

Modelon Impact is a system simulation tool developed by Modelon that supports the Modelica language. all compiled models in Modelon Impact are compliant with FMI standards, and models are created and integrated for FMI, simulation. The main features include

- Supports FMI 2.0.
- Supports Co-Simulation and Model Exchange (export only).
- Windows and Linux (specific commands) are supported.

In addition, when used with the OPTIMICA Compiler Toolkit, which is the kernel of Modelon Impact, it can provide more functionality to support FMI.

- Supports FMI 1.0 and 2.0.
- Supports Co-Simulation and Model Exchange.
- Windows and Linux are supported.

1.9.1. Creating FMUs

To generate an FMU from a model in the Modelon Impact workspace, right-click on the model and select Export FMU (Figure 1.9.1).

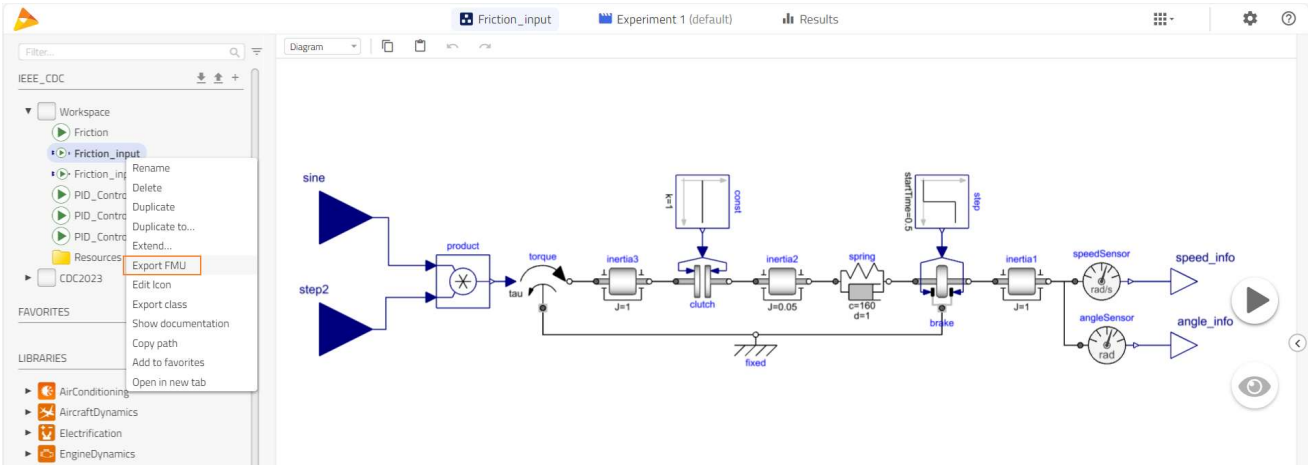


Figure. 1.9 .1 FMU generation

The next screen (Figure1.9.2) displays options for generating FMUs. Select the type of FMU (Model Exchange only, Co-simulation only); if you select a Co-simulation FMU, the solver settings will be the solver state at the time the command is executed.

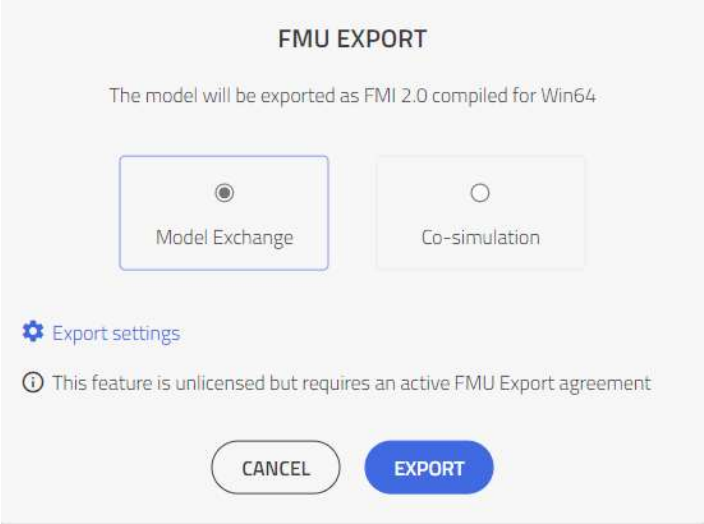


Figure1.9.2　FMU Export Settings

Export settings in Figure 1.9.3 allow you to set other options.

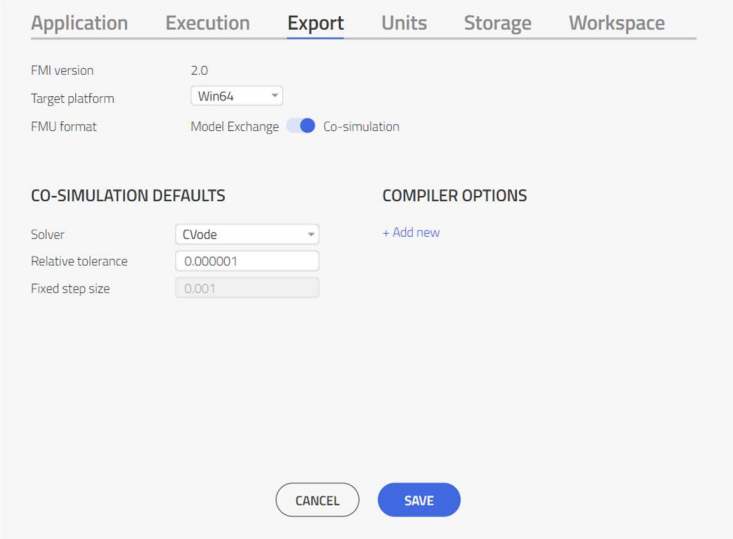


Figure1.9.3　Advanced settings for FMU export

1.9.2.　FMU import and execution

To import an FMU, import the FMU model (Co-simulation 2.0 only) by means of the Import button in the upper left workspace browser (Figure 1.9.4).

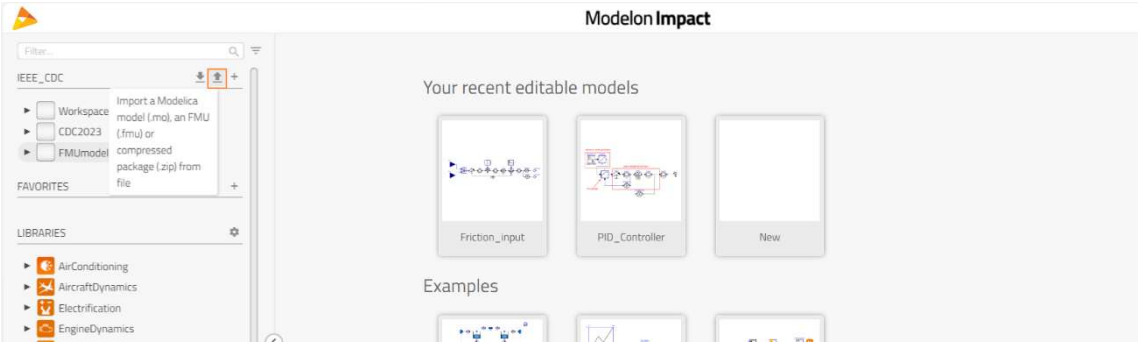


Figure1.9.4　FMU Import

Select the name of the package you want to import and import the model (Figure 1.9.5).

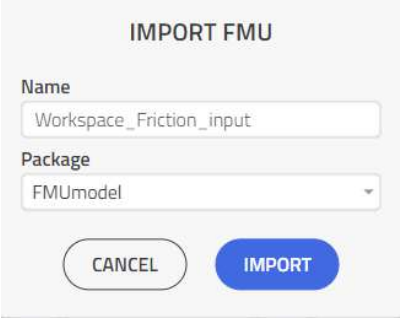


Figure1.9.5 FMU import settings

To run the FMU, drag and drop the imported model onto the model canvas to add it (Figure 1.9.6).

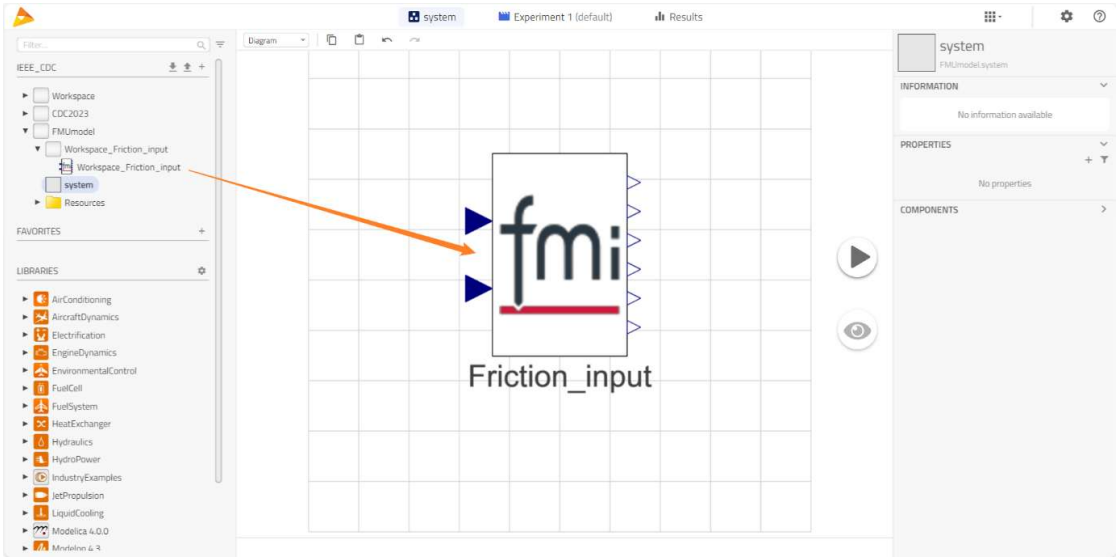


Figure1.9.6 Example model using FMUs

Double-clicking on a component brings up a screen to set parameters; the INFORMATION tab shows information about the model; the PROPERTIES tab is where you set the parameters; the parameters are categorized by tab, and the parameters for the FMU model are displayed. The parameters are categorized by tab, and the model parameters and FMU model parameters are displayed (seeFigure 1.9.7).

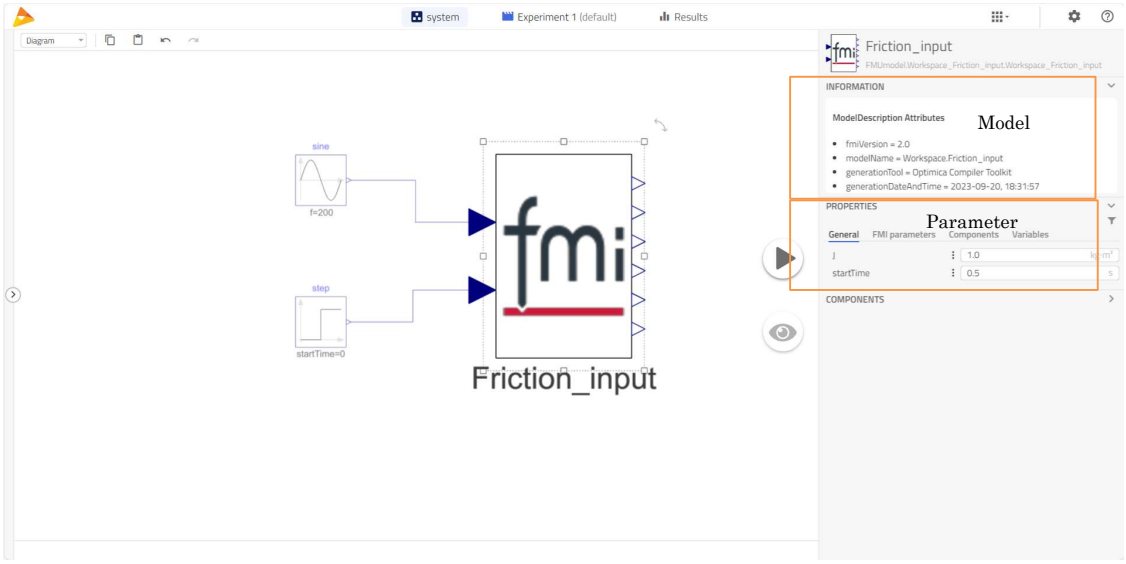


Figure 1.9.7 FMU parameter settings

Imported FMUs can be combined with other FMUs and Modelica components to run simulations in Modelon Impact for results display and analysis.

Multiple FMU modules can also be combined to create a system model.Figure1.9.8 shows an example.

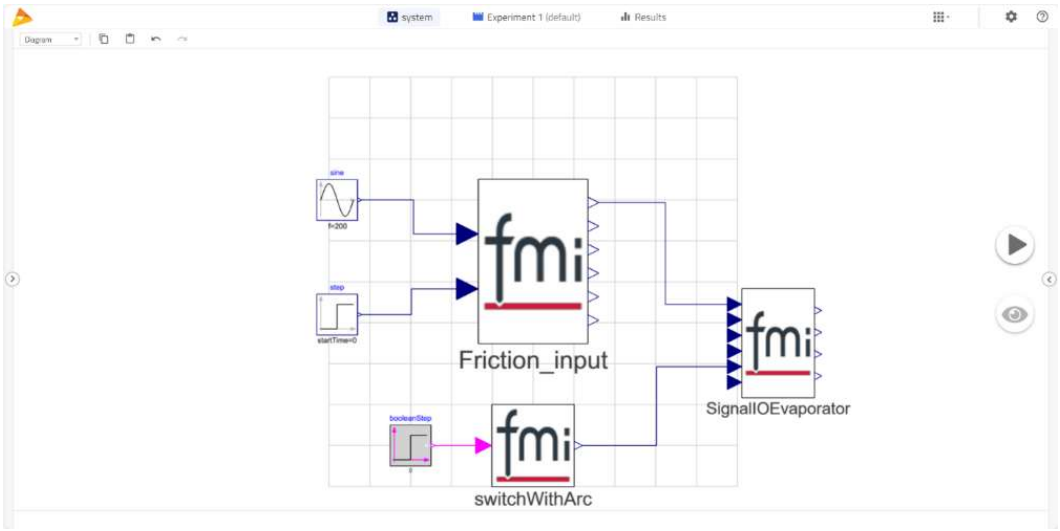


Figure1.9.8　Example of a system model consisting of multiple FMUs

1.9.3.　FMI support function of OPTIMICA Compiler Toolkit

The OPTIMICA Compiler Toolkit, or OCT for short, is the computation engine used by Modelon Impact; it includes the Modelica compiler and solver, and provides functionality beyond dynamic simulation, such as optimization and steady-state calculations. Modelon Impact integrates OCT with Jupyter Notebook as the kernel for FMU model analysis and results analysis.

- Supports FMI 1.0 and FMI 2.0.
- Supports Co-Simulation and Model Exchange.
- Windows and Linux are supported.

Access *JupyterLab* from the Modelon Impact UI and create a notebook (Figure1.9.9).

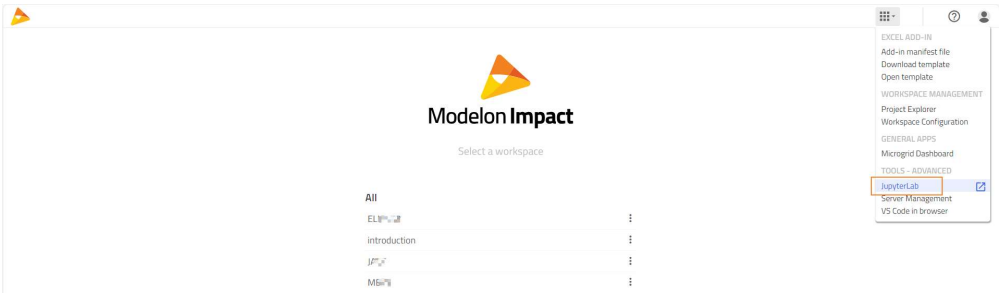


Figure1.9.9 Calling JupyterLab

The notebook is connected to Modelon Impact to read the models in the workspace and perform model analysis from the Python code.

（a） It works with the Modelon Impact workspace and reads the model to be analyzed.

```
from modelon.impact.client import Client

client = Client(url="https://impact.modelon.cloud/", interactive=True)

workspaceName = 'NotebookDemo'
directCapacitor = "Modelica.Electrical.Analog.Examples.Utilities.DirectCapacitor"
inverseCapacitor = "Modelica.Electrical.Analog.Examples.Utilities.InverseCapacitor"
```

```
workspace = client.get_workspace(workspaceName)
model_direct_capacitor = workspace.get_model(directCapacitor)
model_inverse_capacitor = workspace.get_model(inverseCapacitor)
```

（b） Generate FMU model. Here you set the type of FMU model. The default is Model Exchange type 2.0. The FMU models that can be generated are 1.0 or 2.0 version. You can then choose Modelon Exchange, Co-simulation, or ME+CS type.

```
dynamic = workspace.get_custom_function("dynamic")
compiler_opt = dynamic.get_compiler_options()

direct_fmu = model_direct_capacitor.compile(compiler_options=compiler_opt, fmi_target='me+cs', platform='win64', fmi_version='2.0').wait()
inverse_fmu = model_inverse_capacitor.compile(compiler_options=compiler_opt, fmi_target='cs', platform='linux64').wait()
```

（c） Export the generated FMU model for analysis.

```
direct_fmu_path = direct_fmu.download('./Resources')
inverse_fmu_path = inverse_fmu.download('./Resources');
```

（d） Connecting the two models. (Co-Simulation workflow)

```
# Load CS FMUs
from pyfmi import load_fmu

cs_direct_model = load_fmu(direct_fmu_path, kind="CS")
cs_inverse_model = load_fmu(inverse_fmu_path, kind="CS")
```

```
from pyfmi.master import Master

# Define model and connection lists
models = [cs_direct_model, cs_inverse_model]
connections = [(cs_direct_model, "v", cs_inverse_model, "v"),
               (cs_direct_model, "dv", cs_inverse_model, "dv"),
               (cs_inverse_model, "i", cs_direct_model, "i")]

# Create the simulation master object
coupled_model = Master(models,connections)
```

（e） Calculate and enter import values.

```
import numpy as np
import math
import matplotlib.pyplot as plt

offset = 0.
amplitude = 1.
frequency = 1.5
times = np.linspace(0,1,100)
currents = offset + amplitude*np.sin(2*math.pi*frequency*times)
```

```
in_i_s = np.transpose(np.vstack((times,currents)))
input_object_cs = ((cs_direct_model,'iDrive'), in_i_s)
```

（f） Run the analysis and check the results (see Figure 1.9.10).

```
# Reset model if it is already initialized
coupled_model.reset()

# Set communication interval
opts = coupled_model.simulate_options()
opts["step_size"] = 0.001 # "communication interval"

# Set some parameters
cs_direct_model.set("C", 1)
cs_inverse_model.set("C", 0.5)

# Run simulation
results = coupled_model.simulate(input=input_object_cs,options=opts)
```

```
# Plot capacitor voltages

plt.figure(1)
plt.plot(results[cs_direct_model]['time'],results[cs_direct_model]['capacitor.i'], label="capacitor I - direct")
plt.plot(results[cs_inverse_model]['time'],results[cs_inverse_model]['capacitor.i'], label="capacitor I - inverse")
plt.xlabel("Time [s]")
plt.ylabel("Current [A]")
plt.legend()
plt.show()
```

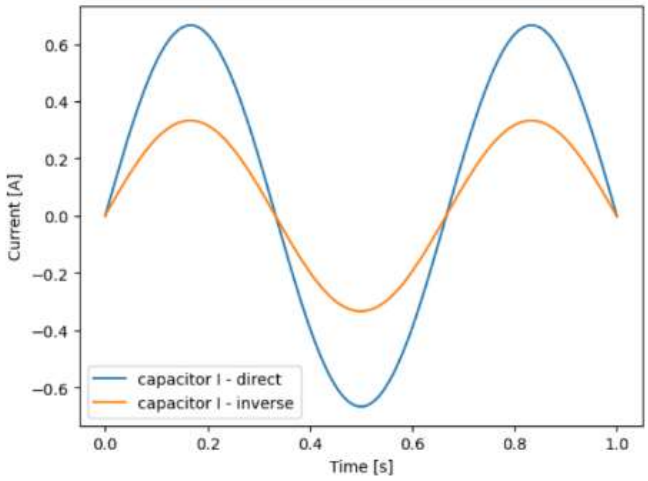
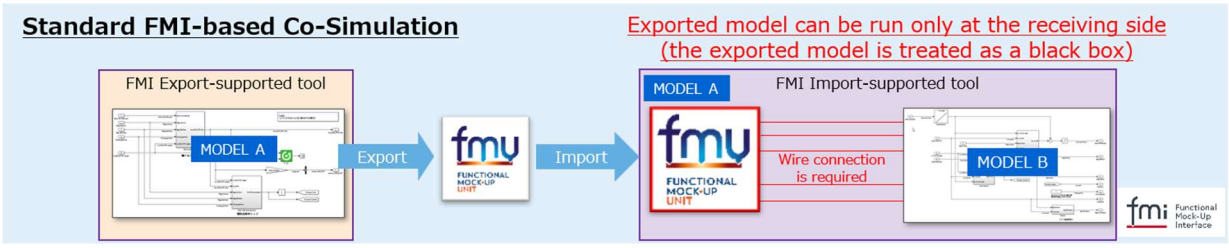


Figure1.9.10　FMU analysis results in Jupyter Notebook

1.10. VenetDCP

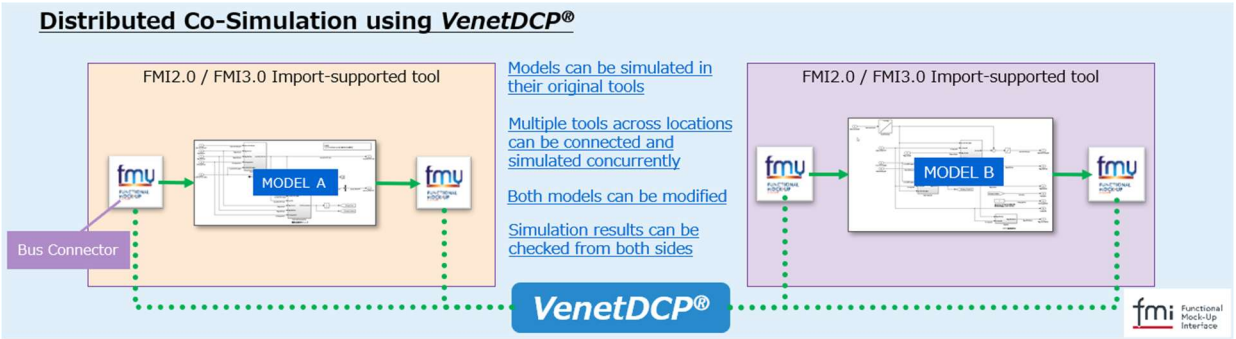
VenetDCP is a Distributed Co-Simulation Platform provided by Toshiba Digital Solutions Corporation that can connect simulation tools of different types and versions.

In a Co-simulation using the usual FMI standard, the model must be exported to FMU and passed to the other party. In this case, the exported model becomes a black box and only the party receiving the model can simulate it.

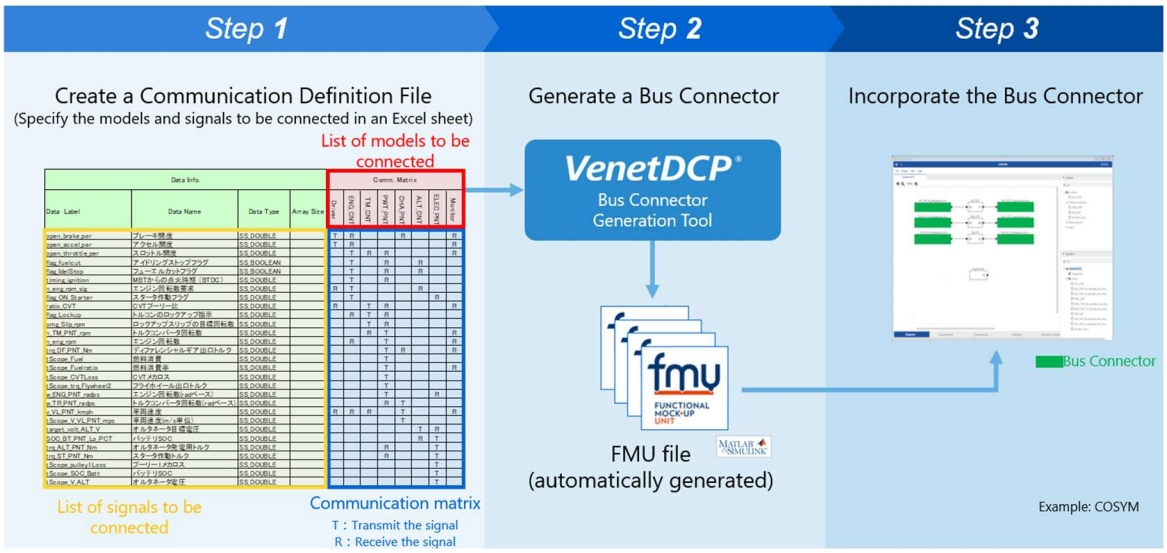


Co-simulation using VenetDCP allows you to simulate a model without passing it on, leaving it in the original simulation tool. In this case, each model must incorporate a "Bus Connector," an FMI communication function block generated by VenetDCP. This Bus Connector is provided as an FMU in FMI 2.0 or FMI 3.0 format and can be used with any simulation tool that supports FMI 2.0 or FMI 3.0 import.

When a simulation is started with a model that incorporates a Bus Connector, that model is connected to other models over the network via the Bus Connector. During simulation, VenetDCP allows each connected model to run concurrently, synchronized in time. This allows for distributed Co-simulations, where multiple simulation tools are linked across multiple locations. In addition, models are not exported to the FMU, so both models can be modified, and results can be checked in both simulation tools because the models are at hand.



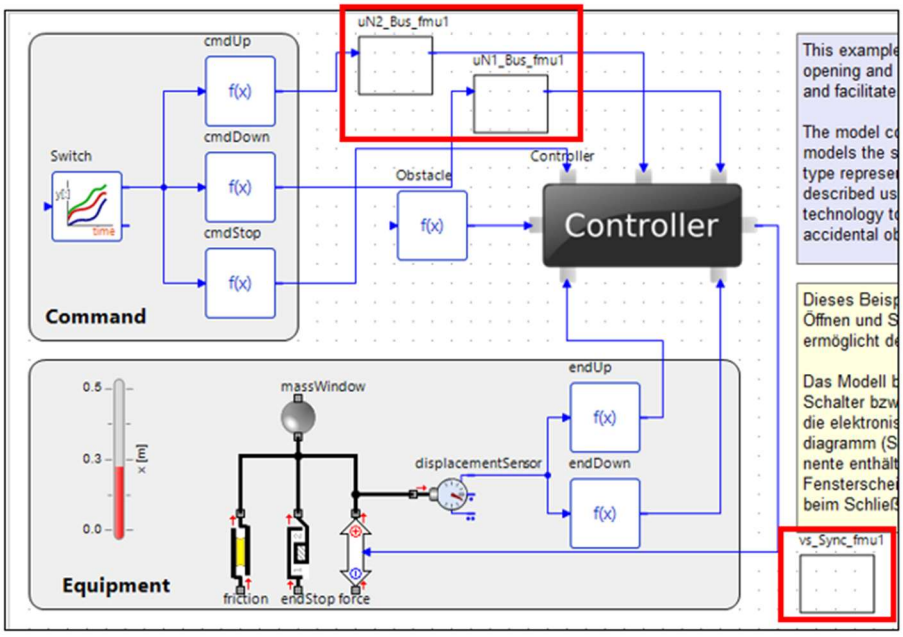
Preparation for distributed Co-simulations using VenetDCP involves the following three steps



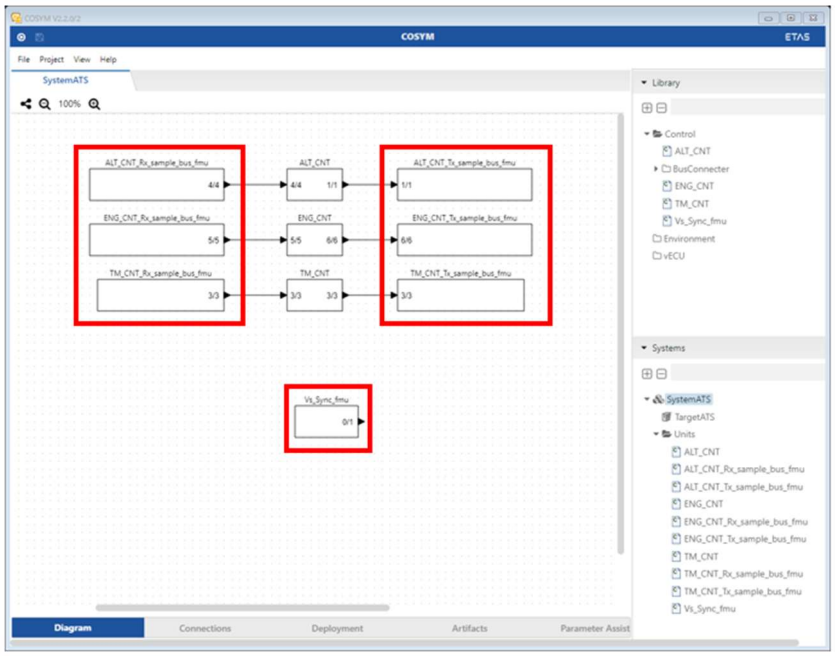
1. Create a "Communication Definition File". The communication definition file specifies the input signals, output signals, and communication matrix (signal connections between models) for each model.
2. Generate a Bus Connector by importing a communication definition file into the "Bus Connector Generation" tool.
   Distribute the generated Bus Connectors to the participants in the distributed and coupled simulation.
3. Incorporate Bus Connectors into the model. Follow the FMI import procedure for each tool.

Below are examples of models incorporating a Bus Connector.

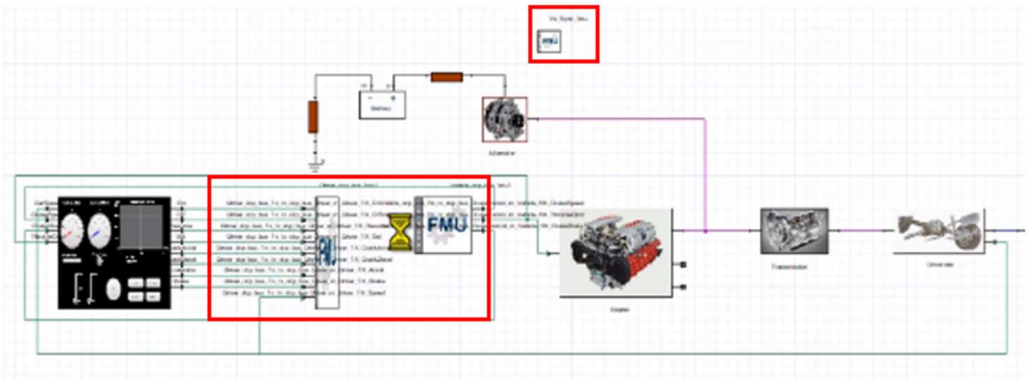
・Connection example 1: Connection with Simulation X



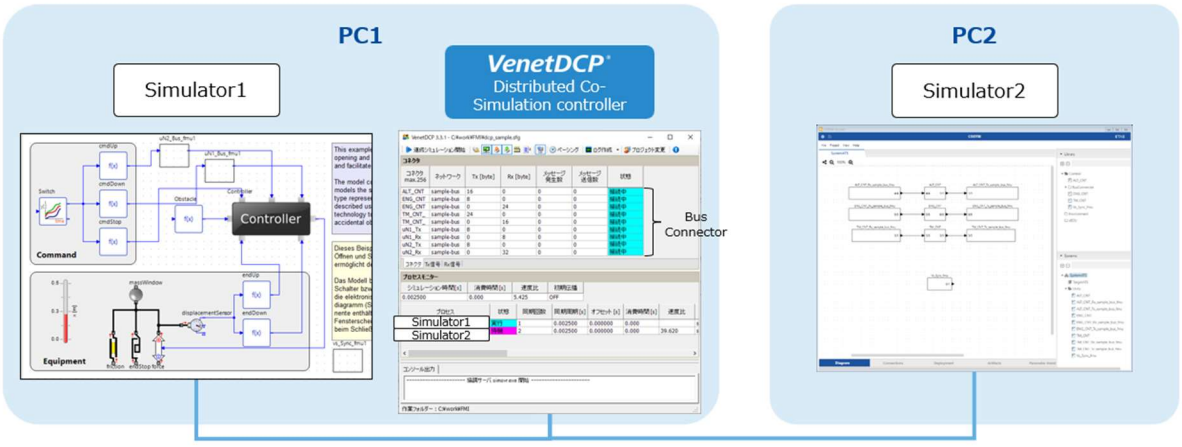
・Connection example 2 ETAS COSYM



・Connection example 3 Ansys Twin Builder



When a simulation is started with a model incorporating Bus Connectors, the Bus Connectors are automatically connected to the Distributed Co-simulation Controller. Once all Bus Connectors are connected, distributed Co-simulation begins.



1.11. OpenModelica

OpenModelica is a simulation tool using the Modelica language developed by the Open Source Modelica Consortium. This tool is available to the public free of charge. The following procedure is based on version 4.2. The features of this version are shown in Table 1.11.1.

Table 1.11.1 List of OpenModelica FMI Interface Functions

FMI Creation Function	
FMI Version	1.0, 2.0
Interface type	Model Exchange, Co-Simulation
OS	Windows, Linux, macOS
License	When creating FMU: Not required
	At runtime of created FMU: Not required
FMI Import function	
FMI Version	2.0
Interface type	Model Exchange
OS	Windows, Linux, macOS
License	When importing FMU: Not required
	At runtime of imported FMU: Depends on the tool used to generate FMU.

1.11.1. Creating FMUs

This section describes how to create an FMU. Settings such as where to save FMUs are required previously.

The save location is specified in the working directory at Tools>Options>General.

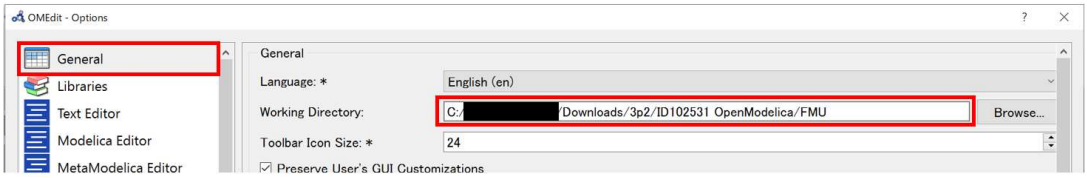


Figure 1.11.1 Setting the working directory with the OMEdit-option

The specification of FMI 1.0 or 2.0, as well as Co-Simulation or Model Exchange, can be configured under Tools>Options>FMI by selecting the export version and type.

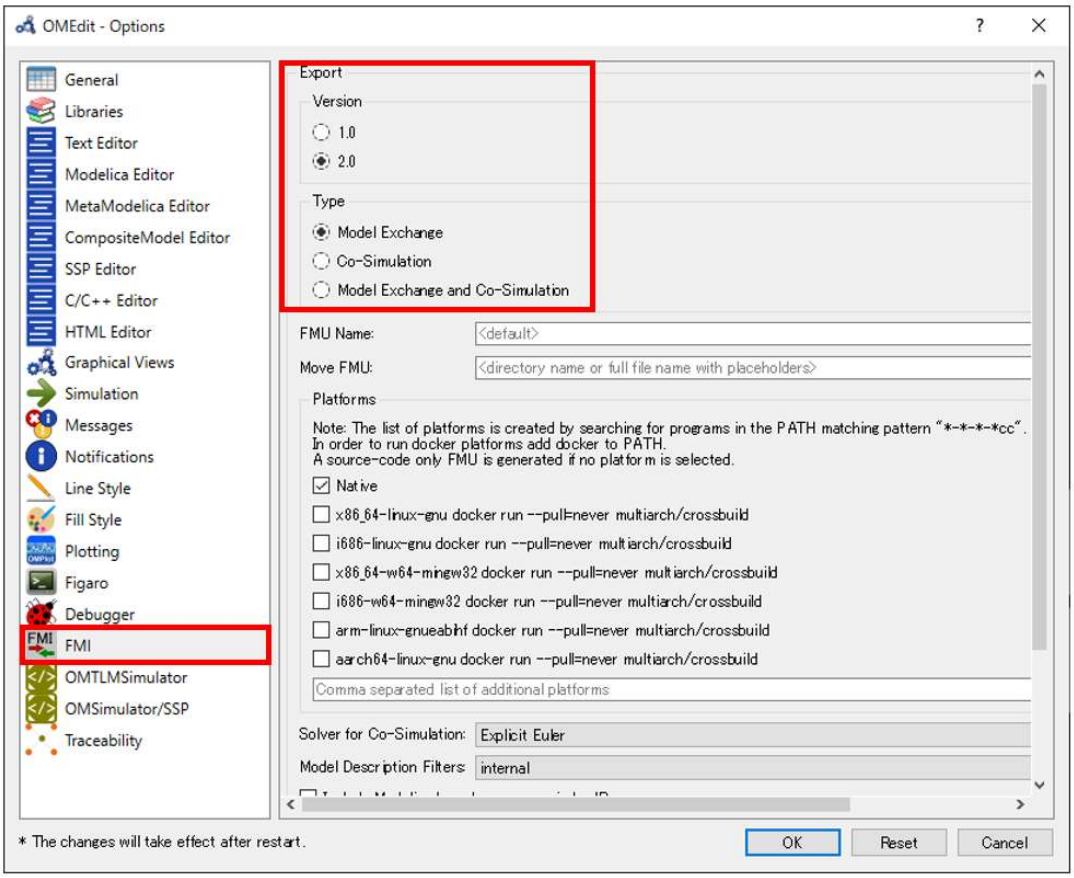


Figure 1.11.2 OMEdit-option to set FMI

Select File>Export>FMU to create an FMU.

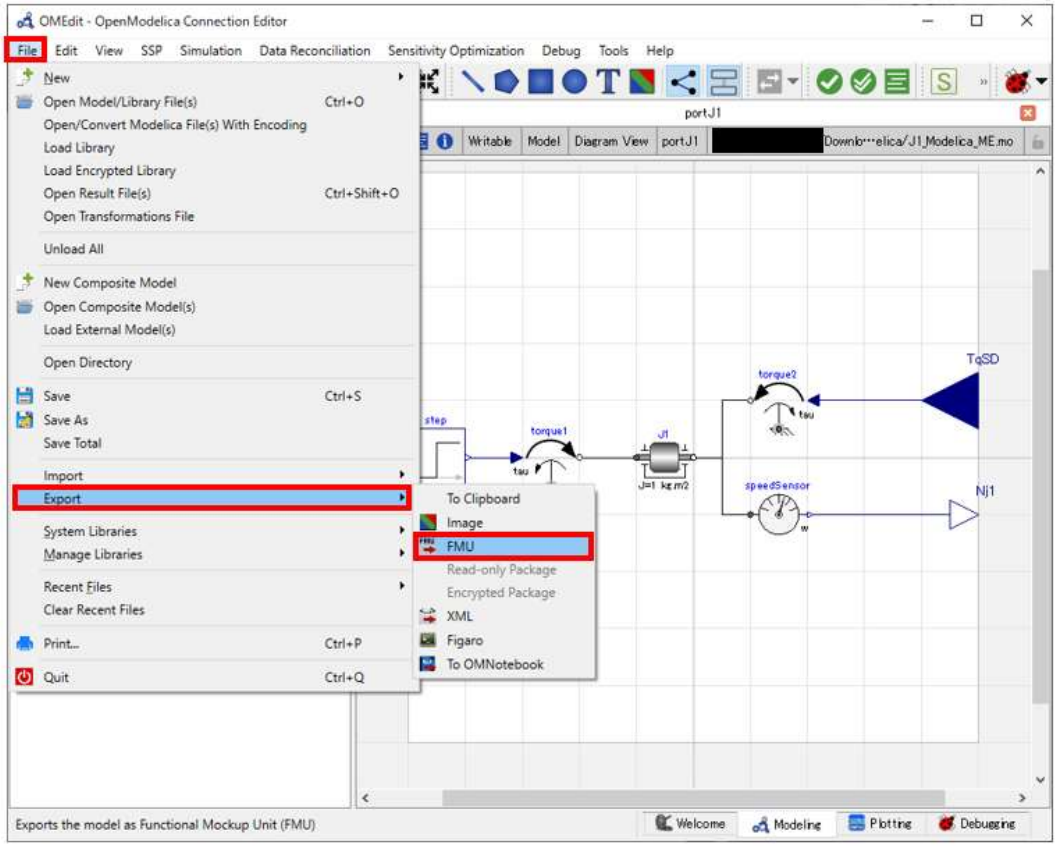


Figure1.11.3 Accessing the FMU

The result of the export operation (success or failure) can be checked in the Messages Browser. In case of success, "The FMU is generated at ～" is displayed.

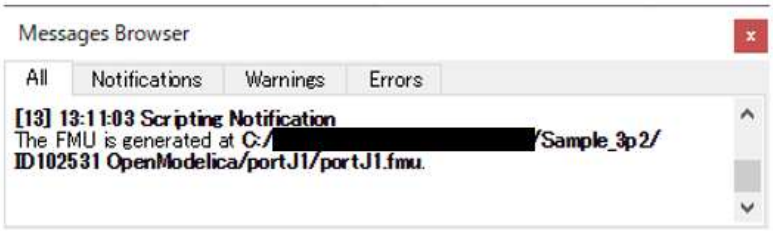


Figure1.11.4 Checking the results of your work in the Message Browser

1.11.2. Importing FMUs

This section introduces the import of FMUs. However, there are many cases where the imported FMU cannot be executed, so this explanation is based on a case where we were able to execute the FMU. Before importing FMUs, go to Tools>Options>Simulation and check "Enable FMU Import" to enable it.

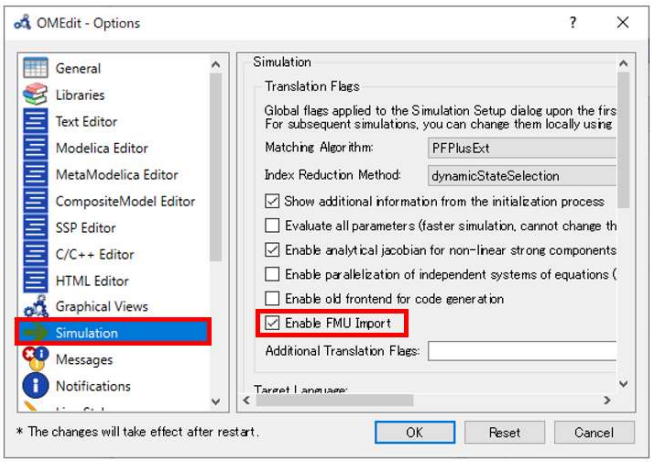


Figure1.11.5 Enable FMU Import

Select File>Import>FMU.

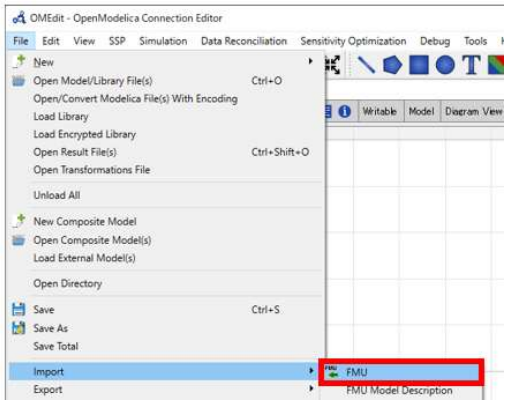


Figure 1.11.6 Accessing the Import Function

Next, specify the .fmu file to be imported.

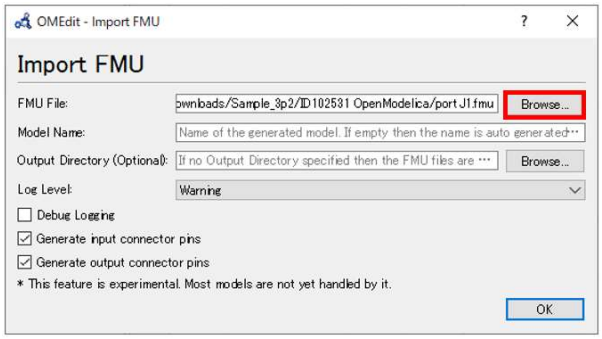


Figure1.11.7 Specifying the .fmu file to be imported

The imported FMU will be shown in the library. You can manually put it in the diagram view.

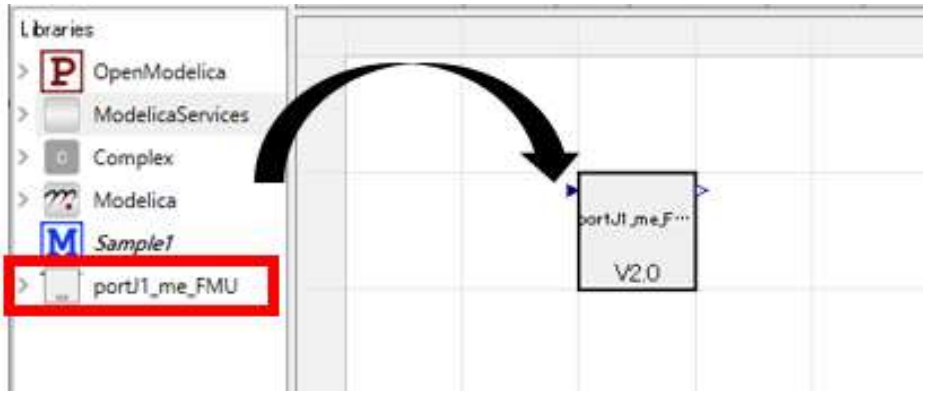


Figure. 1.11 .8　Imported FMUs

1.11.3. FMU Execution

Run a simulation at Modeling>Simulate.

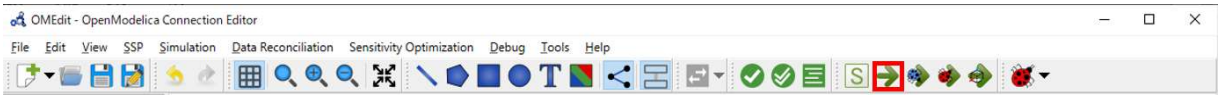


Figure1.11.9 Running the simulation

In the Plotting>Variable Browser, check the variables you wish to observe. Thus, the results for the selected variables are displayed.

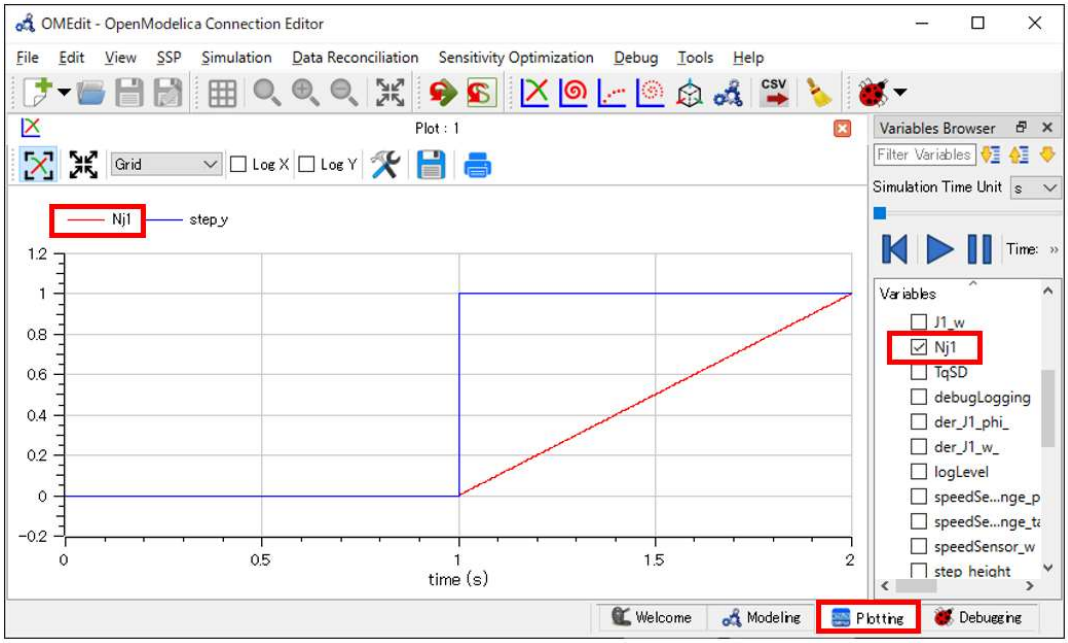


Figure1.11.10 Selecting and displaying output results

1.12. FMPy

FMPy is a free Python library that can simulate Functional Mock-up Units (FMUs) with the following features

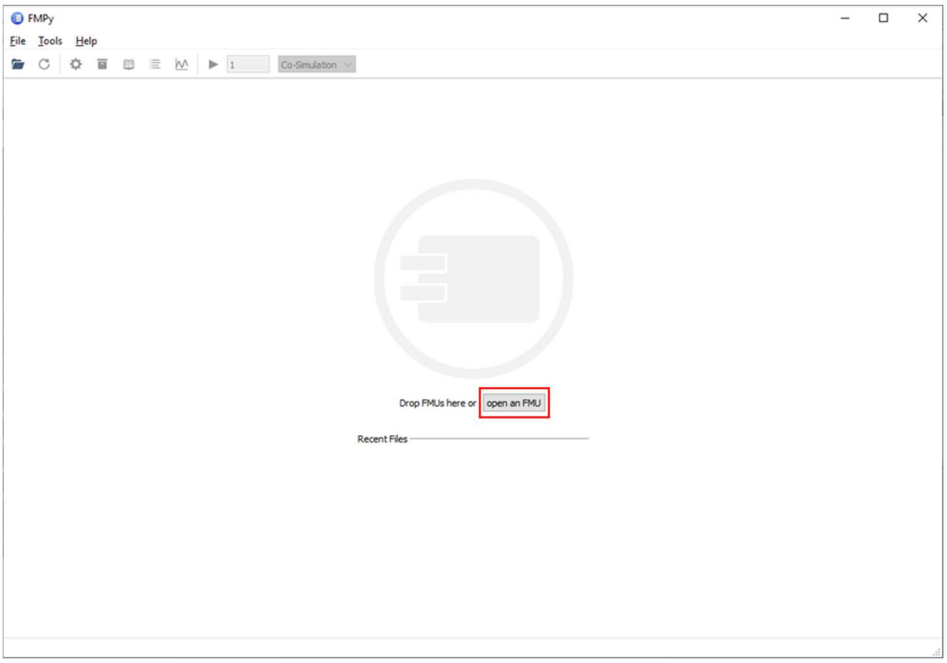
- Supports FMI 1.0, 2.0, and 3.0 (beta).
- Supports Co-Simulation and Model Exchange.
- Can run on Windows, Linux, and macOS.
- command line, graphical user interface, and web applications.
- Jupyter Notebooks can be created.
- You can compile the C code FMU and generate a CMake project for debugging.

FMPy basically only supports the import of single FMUs. If you want to connect multiple FMUs, please use a solution like Dymola.

Let's start the GUI.

```
python -m fmpy.gui
```

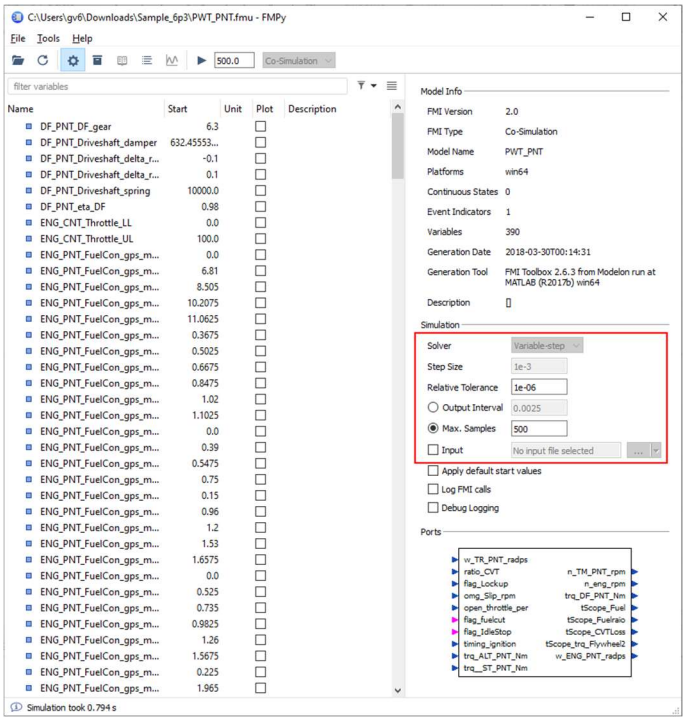
The following screen will appear.



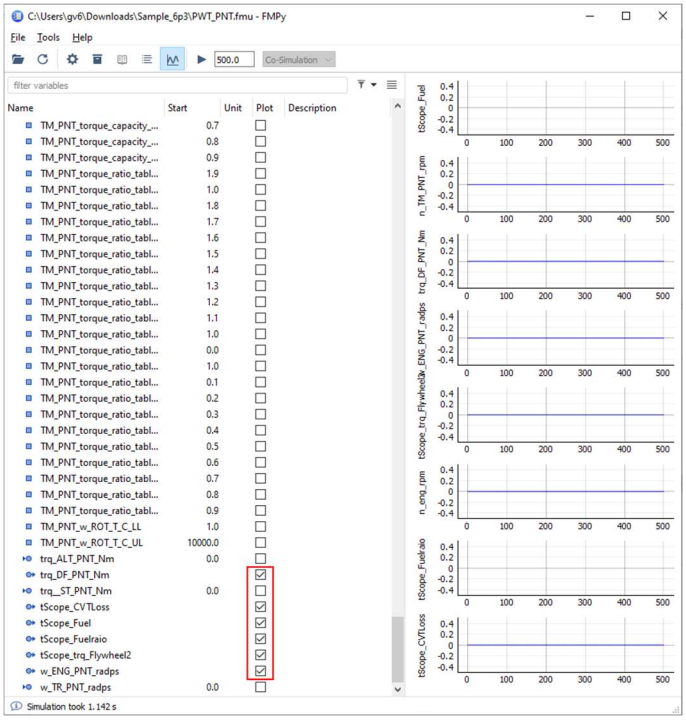
"open an FMU" button or drag and drop to load an FMU.

The simulation period or solver parameters can be specified on the UI.

If you want to use an input file, you can also specify a CSV file by checking the "Input" box on the right side.



Select the variables you wish to view and click the Play button to display the plot screen.



1.13. MAGNA KULI Software

MAGNA's KULI Software is a 1D simulation tool specialized for thermal management in mobility applications. It supports simulation at various stages, from early development phase of cooling system layout to optimization, enabling efficient design. Mobility refers to passenger cars, trucks, trains, construction machinery, agricultural machinery, and other industrial machinery, widely used worldwide.

KULI Software supports creating and loading FMUs. The following summarize the support status in KULI 19.1: 1.13-1 Table: KULI FMI Support Status (as of KULI 19.1)

1.13-2 Table: KULI FMI Support Status (as of KULI 19.1)

FMU Export	
FMI Version	2.0
Format	Co-Simulation (CS)
Required Licenses for Export	Free Version: Not required KULI FMU Pro Edition: Dedicated license required
Required License at export destination	Free Version: All licenses required within the FMU KULI FMU Pro Edition: Not required
Notes	KULI FMU pro version allows encryption and setting FMU usage expiration dates.
FMU Import	
FMI Version	2.0
Format	Co-Simulation (CS)
Required Licenses for Import	A dedicated Advanced License is required.
Notes	KULI connects to the FMU via Python. The intermediary Python script is generated by a dedicated generator, so no coding is required.

1.13.1. Exporting the FMU

Prepare the model to be exported to an FMU beforehand. Configure FMU terminals on the model in advance.

Also, install the FMU Generator for exporting FMUs.

Next, export the model to FMU. As shown in the following Figure -1.13.1-1 , right-click the model you wish to export to FMU within the [project view] tab on the left side of the window, then select [Export as FMU] from the context menu.

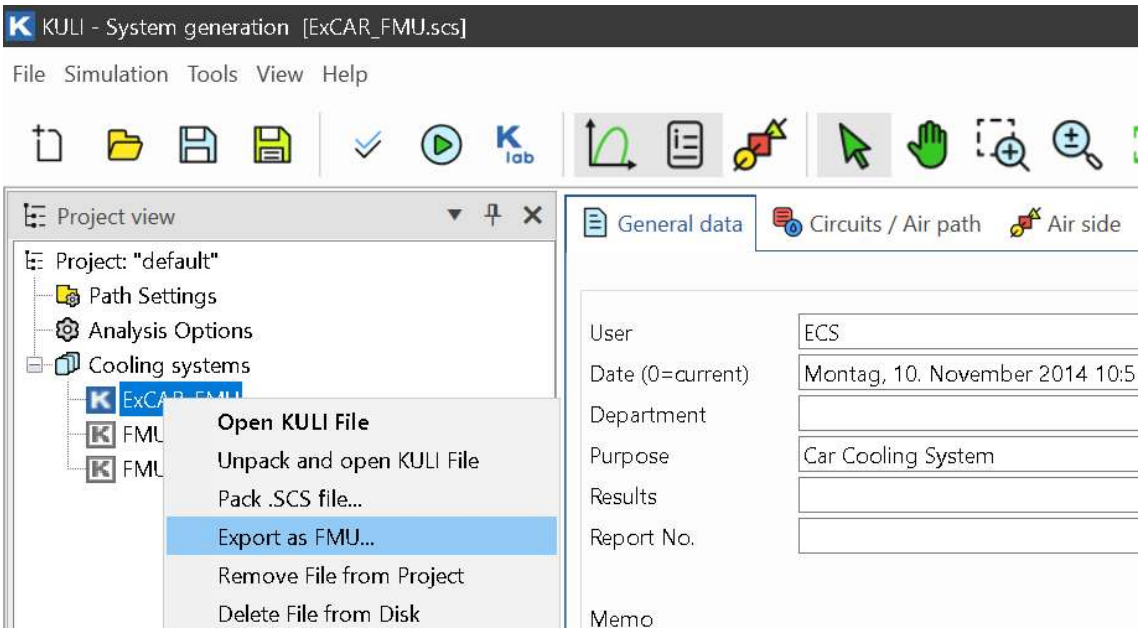


Figure -1.13.1-1 Exporting an FMU from a Model

When executed, the FMU Generator window shown in Figure -1.13.1-2 appears. Configure settings such as the output location, name, expiration date, and encryption status. Press the [Generate] button at the bottom of the screen to generate the FMU.

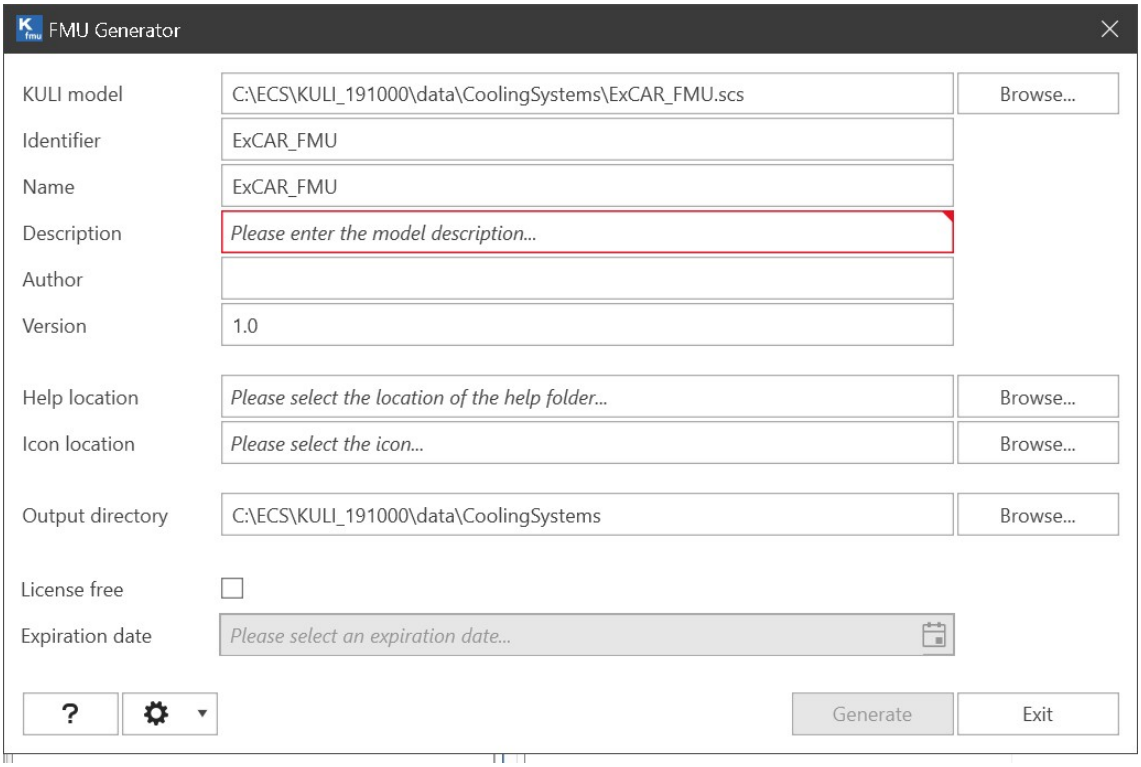


Figure -1.13.1-2 FMU Generator Window

1.13.2. Importing and Running FMUs

KULI uses Python for connecting with FMUs. By creating Python scripts as APIs and specifying them in KULI, co-simulation becomes possible. Python scripts can be generated using a dedicated generator. This generator is operated solely via GUI, requiring no coding.

This section explains how to use the Python script generator and configure Python scripts in KULI.

First, please install KULI_FMU_importer beforehand. For details, please contact support.

After installation, running Start_Script_Generator.bat launches a GUI shown as: Figure: 1.13.2-1. Operate the GUI to specify the FMU file, configure General data (like time step), set terminals, and designate the script output location. Press the [Generate script] button to generate the script.

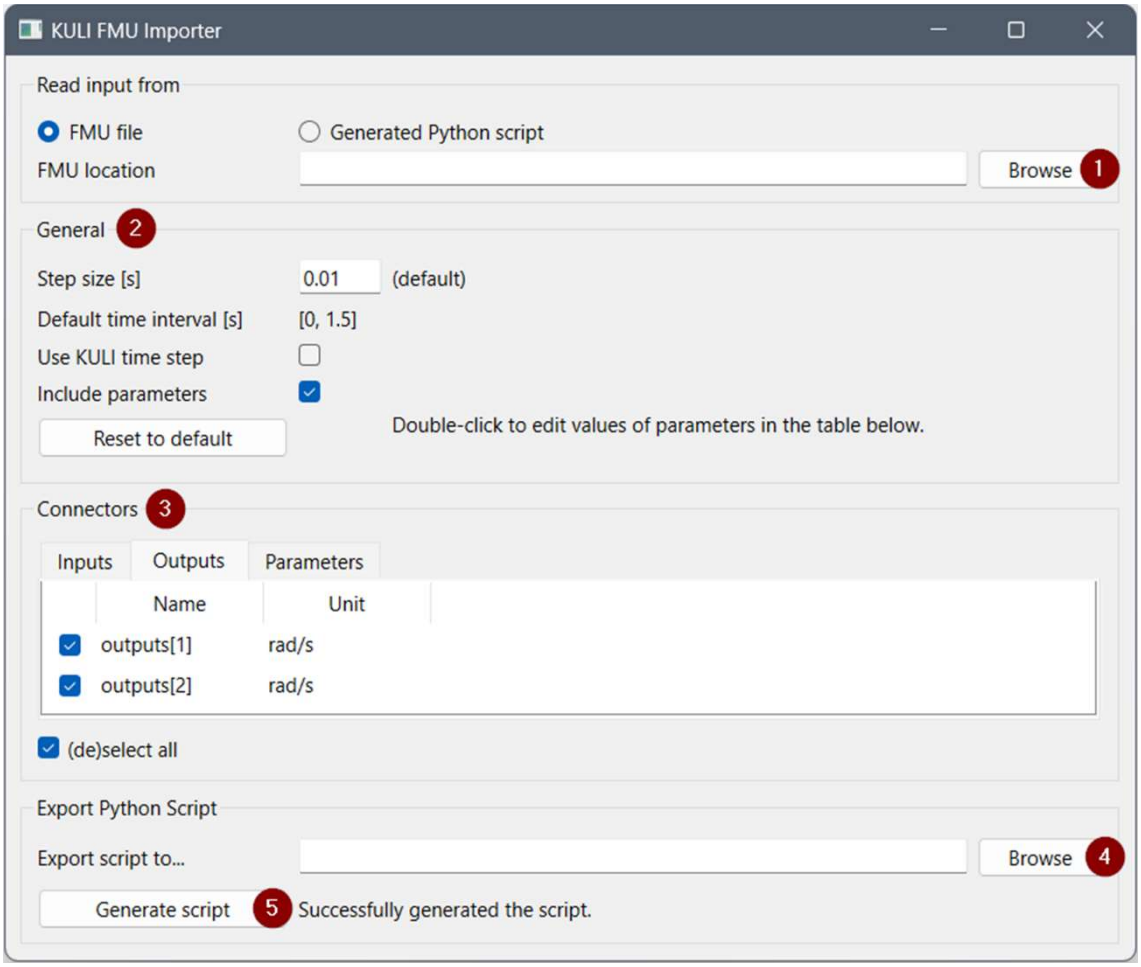


Figure 1.13.2-1 Start_Script_Generator.bat GUI

Apply the Python script you just generated to the model. From this step onward, the operation is performed in KULI. Configure the Python script from the [Python controller] block. Drag the [Python controller] block into the circuit area, then double-click the block to open its edit window. Figure 1.13.2-2 The GUI window shown in the figure will open. Specify the Python script, select the Function, and configure boundary terminals with the FMU in the GUI. Press [OK] to complete the editing.

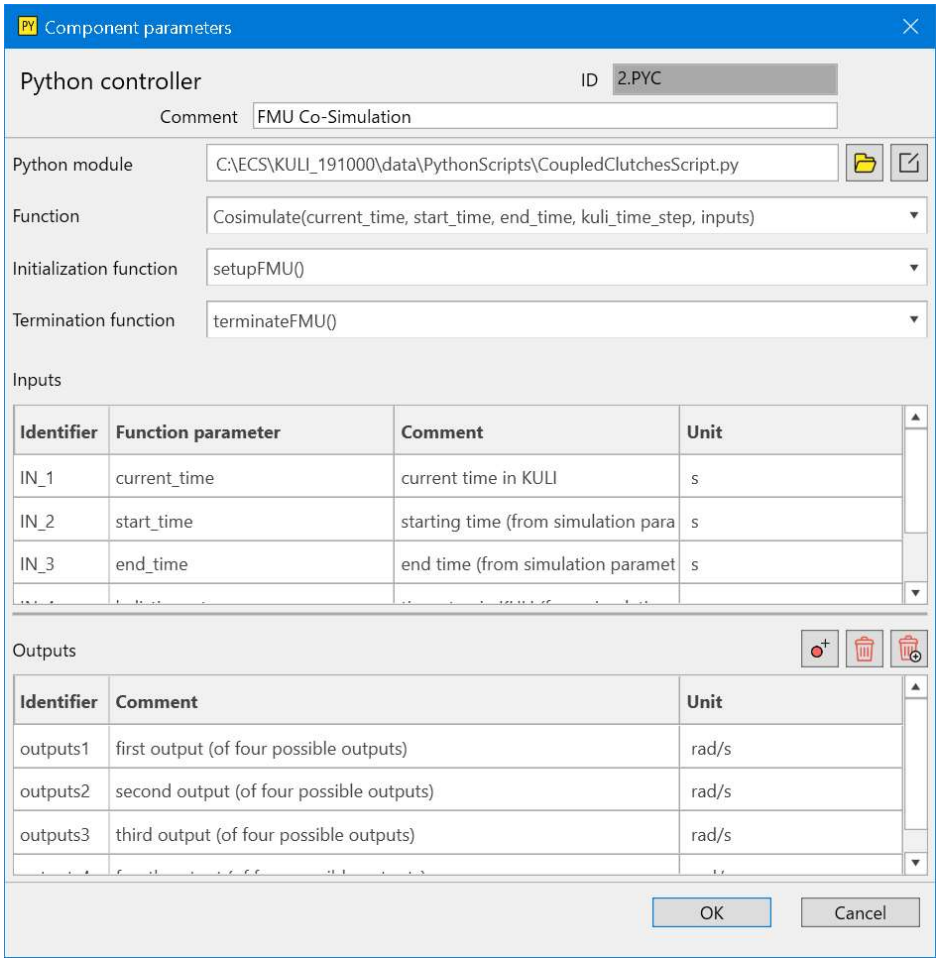


Figure 1.13.2-2 Python controller block　Edit Window

Connect the FMU block (Python controller) to the KULI block as shown in Figure1.13.2-3 to complete the model. Pressing the [Start simulation] button at the top of the window executes the calculation, providing fast and accurate results.

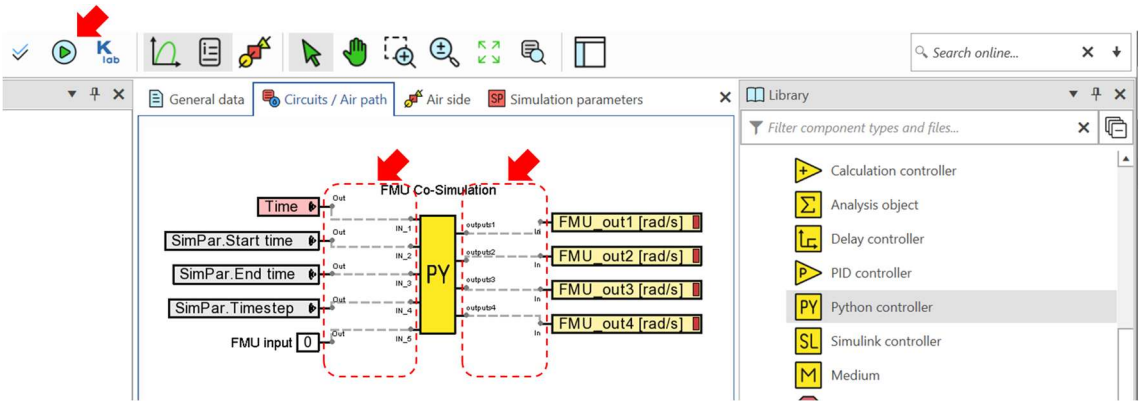


Figure -1.13.2-3 KULI model with connected FMU

Chapter 2: FMI Application Examples

This chapter provides examples of FMI applications

2.1. FMI utilization model for hybrid aircraft

The tutorial will use a hybrid aircraft model created by Nagoya University and Mitsubishi Heavy Industries Aeroengine Corporation as an example, as part of the "Aichi Center of Knowledge Aichi Priority Research Project" for 2021.

2.1.1. Description of Sample Models

The aircraft model consists of a battery model created in the master tool (Simplorer) and two FMUs: an electric drive model (Modelica) and a hydraulic system and dynamics model (Amesim) for model exchange. Figure 2.1.1.

The OS is Windows 64-bit version and FMI2.0 only.

However, please note that a sample model of this model cannot be released due to rights reasons.

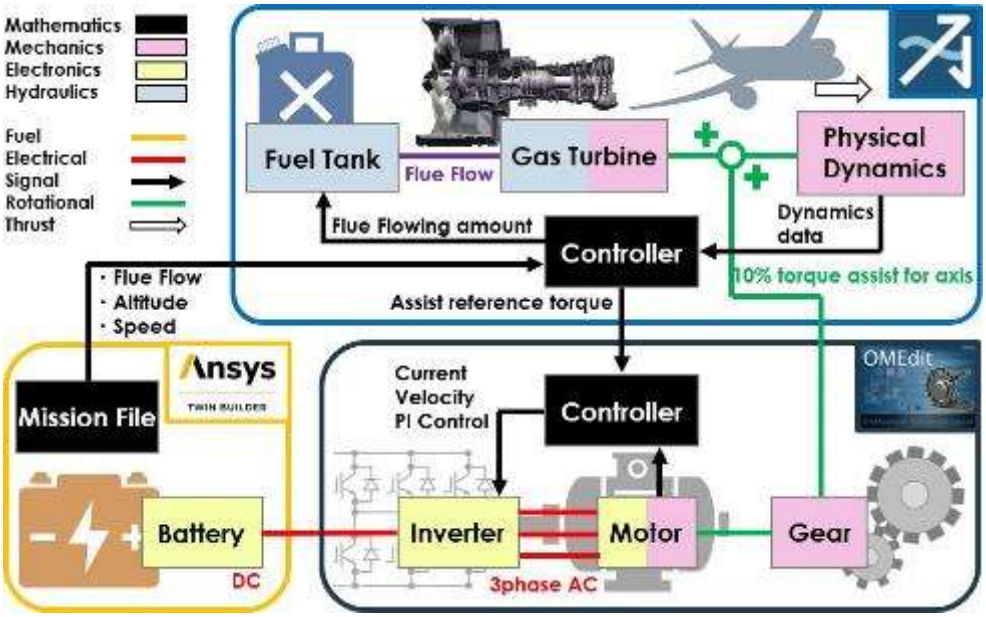


Figure2.1.1 Hybrid aircraft's system simulation model

Simplorer was used as the master tool. Two FMUs are imported into Simplorer. The importing method varies depending on the tool, such as specifying on the menu screen or dragging and dropping from Explorer, so please refer to the manual.

In addition, the master tool provides the flight data of the battery model and aircraftshows the flight data of the battery model and aircraft (Mission File in Figure 2.1.1).

2.1.2. Model Overview

2.1.2.1. Electrical System Modeling

In the electric system modeling, the DC voltage received from the battery is converted to 3-phase AC by the inverter, and the process of outputting torque by the motor is represented by a mathematical model.Figure2.1.2 Figure 2.1.2 shows a block diagram of the electrical system model. This section describes the individual losses of the battery, inverter, motor, controller, and power semiconductors.

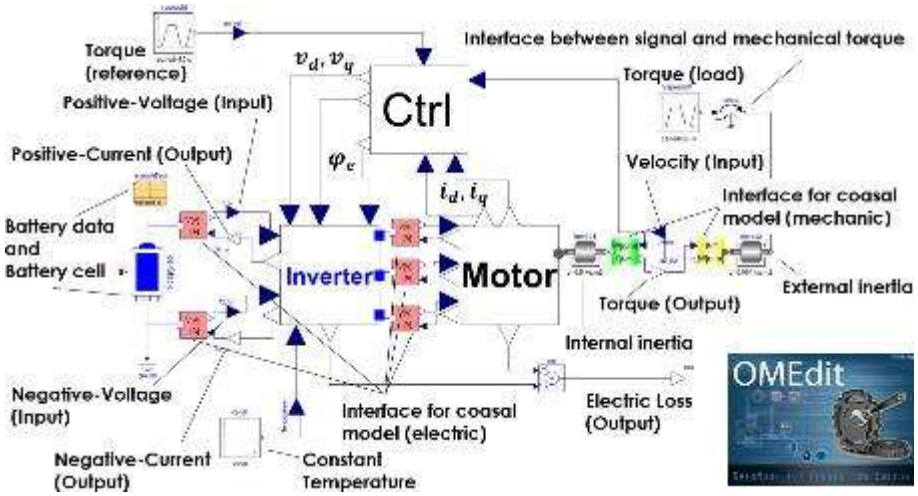


Figure2.1.2 Electrical Drive Block Model

<1: Battery>

For the battery, we used a model that represents the lithium-ion battery, which has the highest energy density of all rechargeable batteries in practical use, as an electrical equivalent circuit.

<2: Inverter>

Inverters originally perform DC/AC conversion using switching. In this study, however, the averaging method is used to reduce computational cost, as shown in Equation (1). However, the values ofV_{DC}, I_{DC} are the DC voltage and current at the inverter input, respectively, andv_{ac}, i_{ac} denote the output three-phase AC voltage and current, respectively. and P_{inv} indicates the losses generated by the inverter.Figure 2.1 .3 Figure 2.1.3 shows the voltage waveforms when the averaging method and switching are considered. The averaging method assumes an ideal output without considering the effects of actual switching and calculates the inverter losses (P_{inv}) is calculated separately. The switching loss and conduction loss considered in this study are described in detail in <5: Power semiconductors>.

$$V_{DC}I_{DC} = i_{ac}v_{ac} + P_{inv} \quad (1)$$

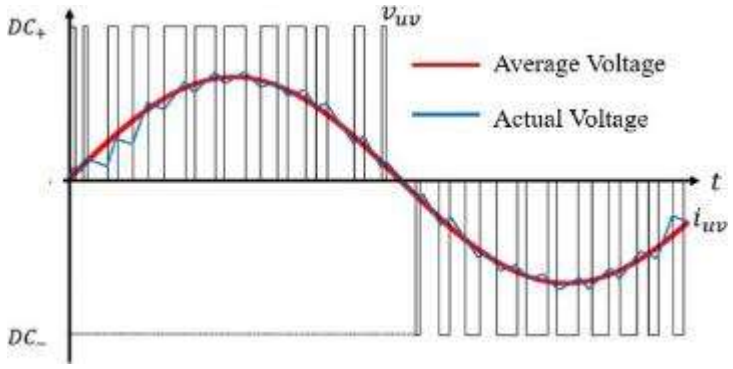


Figure 2.1 .3 Model averaging

<3: Motor>

The motor is an embedded magnet synchronous motor (IPMSM), an AC motor that converts three-phase AC power received from an inverter into torque. This motor features high torque and high speed rotation, and is used in electric vehicles. Equation (2) shows the voltage equation on the dq axis, and equation (3) shows the torque (τ_e) obtained by IPMSM is shown in Equation (3). Note thatv_d, v_q is the dq-axis voltage,R_a is the armature resistance, andi_d, i_q is the dq-axis current, andL_d, L_q is the dq-axis inductance, andω_e is *the* electric angular velocity, $and K_E$ is the chain flux of the motor, and$p$ is the number of poles.

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = R_a \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} -\omega_e L_q \\ \omega_e L_d i_d + \omega_e \Psi_a \end{bmatrix} \tag{2}$$

$$\tau_e = \frac{p}{2}\{K_E + (L_d - L_q) i_d\} i_q \tag{3}$$

<4: Controller>

The controller receives the command torque, selects the optimal control from the rotation speed, current flowing in the motor, voltage applied to the motor, and parameter settings, and outputs the command voltage to the inverter using non-interfering current control. The command value (input) to the controller is set at one-tenth of the output torque required for the aircraft to achieve the specified flight. The analysis was performed as a parallel hybrid system in which the motor assists the gas turbine output. Since IPMSM is employed in this study, the flowchart in Figure 2.1.4 is proposed as the control scheme.

Figure 2.1.5 shows the specific operating points for each control. The horizontal axis shows the d-axis current (i_d) and the vertical axis shows the q-axis current (i_q).

The maximum torque/current control shown in Figure 2.1.5(a) is based on an algorithm that selects the minimum current $|i_{dq}| = \sqrt{i_d^2 + i_q^2}$ to achieve the command torque at low and medium speeds.

In the command torque-voltage limit control shown in Figure 2.1.5(b), (a) When the maximum torque/current control point is out of the control range, an algorithm is constructed to select the intersection of the command torque curve and the voltage limit curve as the minimum current **I** that will achieve the command torque.

For the current and voltage limit control shown in Figure 2.1.5(c), an algorithm was constructed to select the intersection of the current and voltage limit curves as the current operating point that achieves the maximum torque when the command torque is outside the controllable range of the

current.

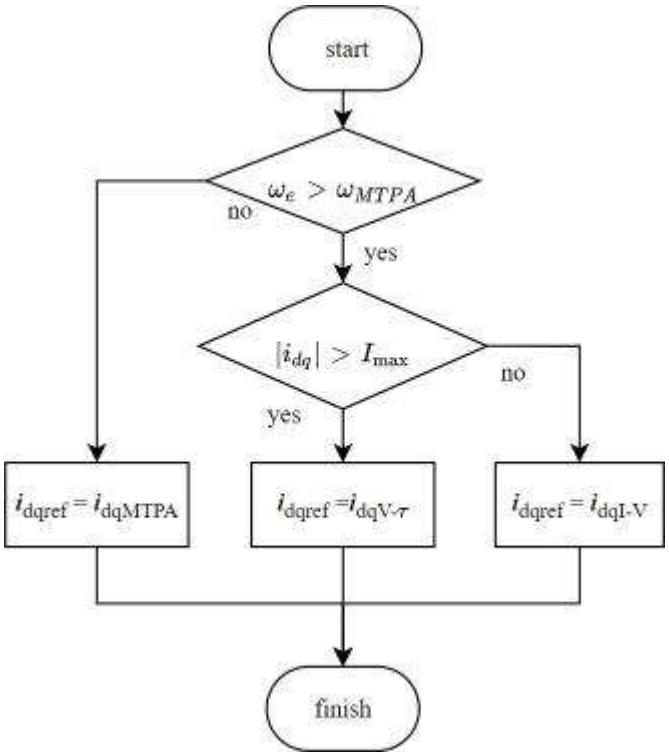


Figure 2.1 .4 IPMSM control flow chert

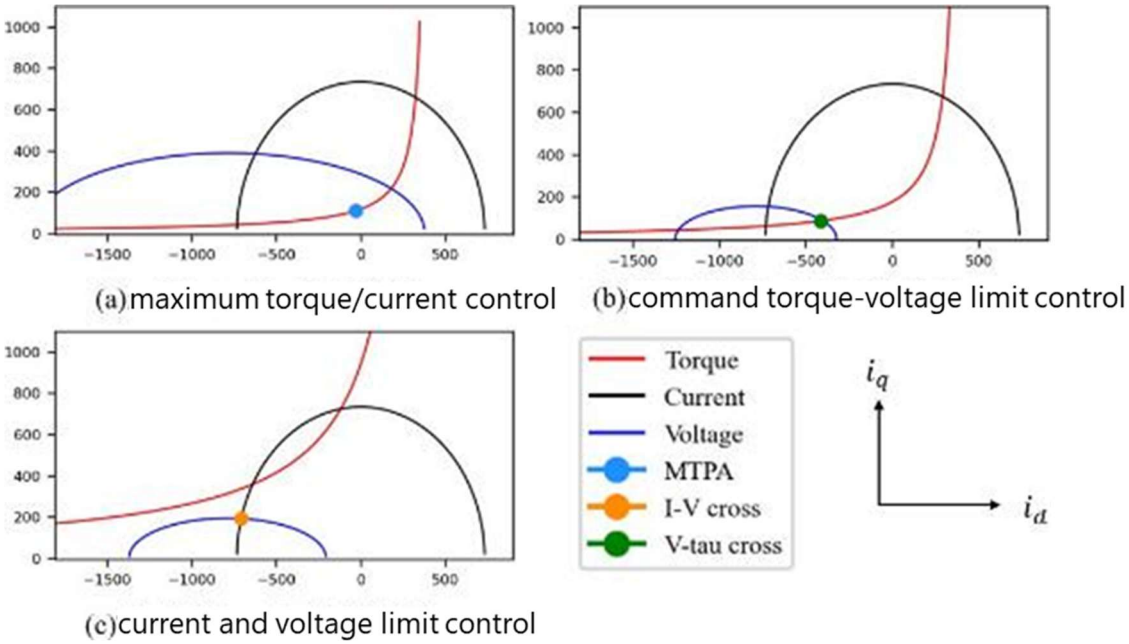


Figure2.1.5 IPMSM control patterns

<5: Power semiconductor loss>

The power semiconductor loss (P_{sem}) of a power semiconductor is the conduction loss (P_{con}) and switching loss (P_{sw}) as shown in Equation 4. First, conduction loss is defined as the sum of the current flowing in the power semiconductor (I_{ds}) and on-resistance (R_{on}) of the power semiconductor as shown in Equation 5. Next, the switching loss is calculated separately from the DC/AC conversion and referenced as a table, as described in <2: Inverter>.

The details are described below. First, the parameters specific to the power semiconductor are obtained from the datasheet. Then, it determines if a loss table exists for this power semiconductor. If there is no loss table, the double-pulse test circuit shown in Figure 2.1.6 (a) is analyzed using LT-Spice. Next, a double-pulse test circuit is analyzed using LT-Spice.Figure2.1.6 (b) shows the analysis results. From these results, the losses on the on- and off-side were obtained by integrating.

Based on the above, we have created a table (Table(V_{ds}, I_{ds}, T_j)) that outputs the power dissipation with the drain-source voltage (V_{ds}), current (I_{ds}), and junction temperature (T_j) of the power semiconductor as arguments, as shown in Equation 6.

Figure2.1.6 (c) shows the procedure for calculating power semiconductor losses. The advantage of this analysis method is that it does not require model analysis that takes sequential switching into account, thereby reducing analysis time.

$$P_{sem} = P_{con} + P_{sw} \tag{4}$$

$$P_{con} = R_{on} I_{ds}^2 \tag{5}$$

$$P_{sw} = Table(V_{ds}, I_{ds}, T_j) \tag{6}$$

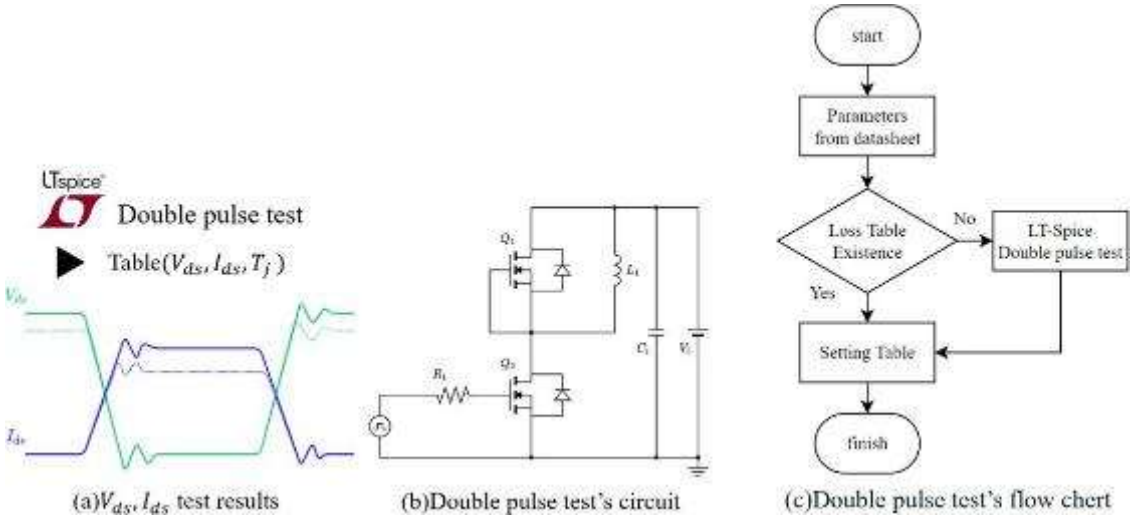


Figure2.1.6 Double pulse test

2.1.2.2. Aircraft and gas turbine system modeling

Amesim was used to model the hydraulic system. An overview is given below. Using the flight mission profile (speed and altitude) as input, the fuel flow rate and command torque were calculated in the control section and used as input to the hybrid engine. In addition, the number of passengers and cruising range are adjusted to make the aircraft system feasible by taking into account the weight of the electrical system, which is simply estimated.

2.1.2.3. Model Connections

Next, Figure 2.1.7 shows the connection relationship of each FMU model in the master tool (Simplorer). Refer to this figure to connect each FMU to the battery and flight data derived from Simplorer. Note that all inputs and outputs are treated as signal signals without units. Also note that a negative value for the rotational speed of the gas turbine indicates rotation in the drive direction.

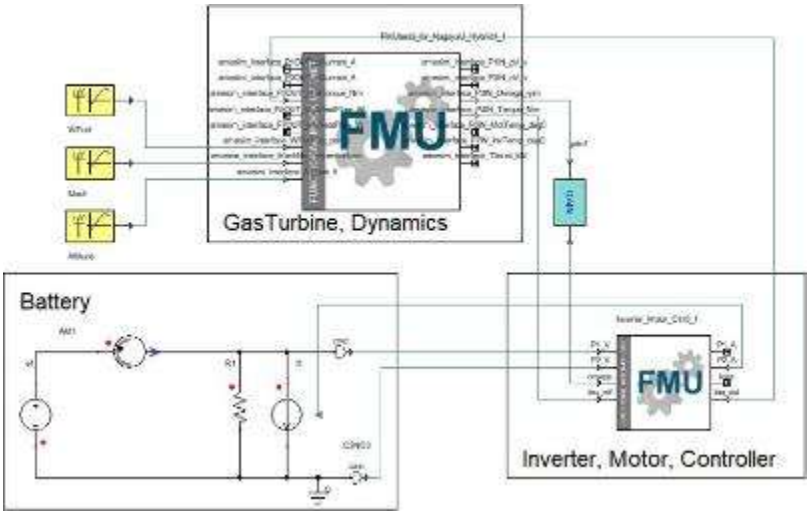


Figure 2.1.7 Model connection (@Master tool)

2.1.3. Aircraft flight data

The flight data is represented by the yellow blocks in the upper left of Figure 2.1.7. These represent altitude, fuel flow rate, and airframe speed, respectively. Figure 2.1.8 shows typical flight data for a medium-sized aircraft with about 100 passengers. This data is used as command values to the controller.

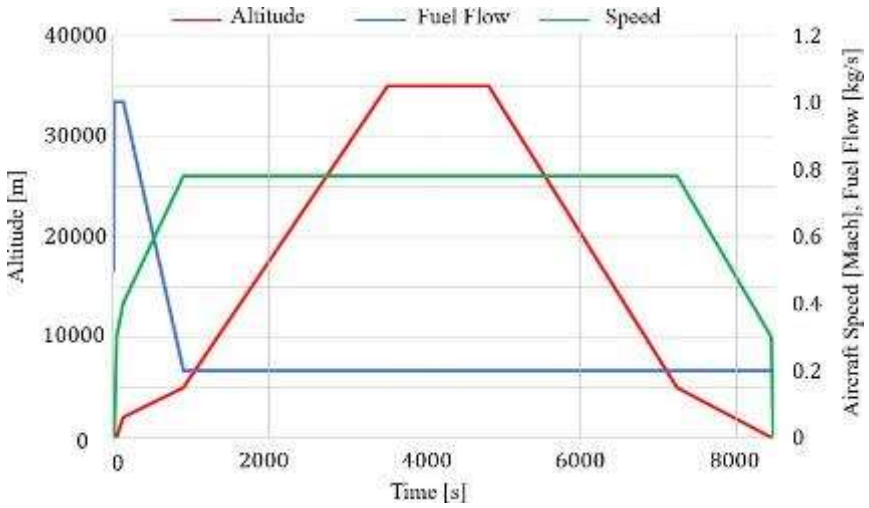


Figure2.1.8 Flight profile (input data)

2.1.4. Master Tool Simulation Settings

Figure 2.1.9 shows the simulation parameter setting screen in Simplorer. The simulation time is set to 8430 seconds.

The step size is variable according to the time constant of the analysis results. The minimum and maximum step size values shown below are default values (Figure 2.1.9). These initial values are set to change as the time constant increases or decreases.

Start Time 0 s

Stop Time 8430 s

Solver and Tolerances TwinBuilder, the 1 ms

Integration method Adaptive Trapezoid-Euler

Min. Calculation Step Size 3.3 μs

Max. Calculation Step Size 10 ms

Min.Output Step Size 0.01 s (arbitrary, since this is the sample time of the output result)

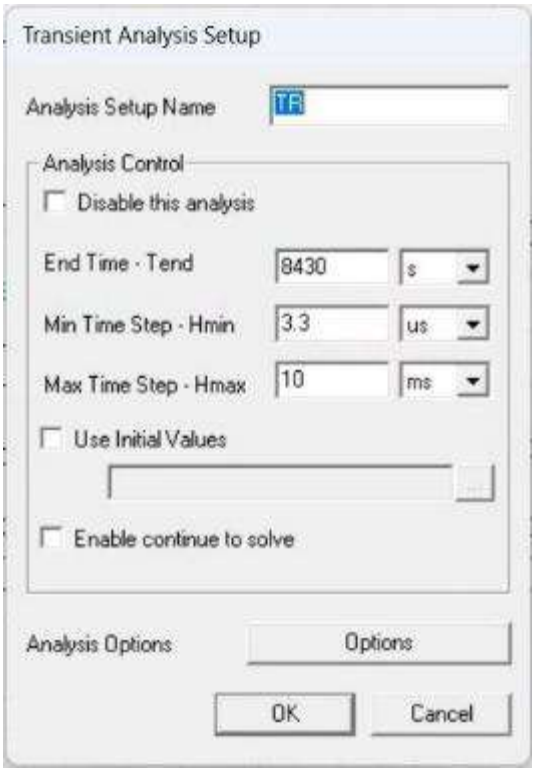


Figure2.1.9 Simulation parameters setting Simulation results

The results of the analysis of this model are shown in Figure 2.1.10. This analysis was performed for a flight of8430 s The analysis was performed for a flight of In this analysis, the torque of the aircraft engine is assisted by a motor.10 % The parallel hybrid system, in which the torque of the aircraft engine is assisted by the motor, is used from the start of takeoff, when fuel consumption is high, to the end of the flight. 1200 s The results show that the motor assists from the start of takeoff, when fuel consumption is high, to the end of the flight. where ω_e is the rotation speed of the motor [krad/s], H is the altitude [kft], T is the thrust applied to the aircraft [kN], andτ_e is the output torque of the motor [kNm], andv_M is the speed (speed of sound)[- (dimensionless)].

Figure2.1.10 shows that the thrust applied to the aircraft is very high from the start of the analysis to1200 s It can be confirmed that the thrust applied to the airframe is very high from the start of the analysis to the

The torque produced by the gas turbine in that section is called the "torque". The torque produced by the gas turbine in that section is10 % The motor torque (τ_e) assists the gas turbine.
The results of this analysis were obtained for a flight mission profile (speed and altitude) for a regional or mid-size aircraft, and even taking into account the weight of the electric drive mechanism, the motor assist extended the range of the aircraft. The results were also compared with the flight results of a similarly sized gas turbine-only aircraft.

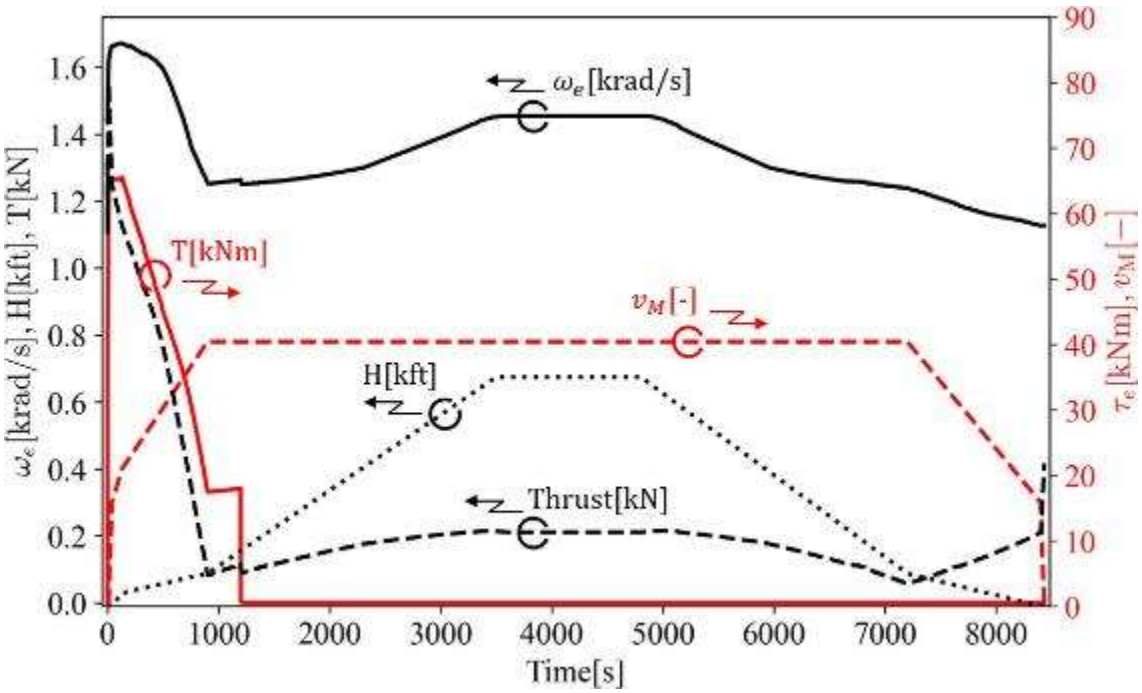


Figure2.1.10 Simulation results

2.2. Coupled 1D-CAE and 3D-CAE

2.2.1. Survey Objectives (2020-2023)

In recent years, 1D simulation models have been used in model exchange processes based on FMI conventions to operationalize models among various tools and attempts have been made to apply them to complex problems. Considering the future distribution of models utilizing FMI, the demand for function extraction and co-simulation in conjunction with 3D simulation models is expected to increase. The objective of the WG was to investigate what kind of information is required for a wide range of applications, to study the methods to realize them, and to extract the effectiveness and issues.

2.2.2. Definition of target tool types and tool trends

Three types of tools were classified as types of target tools ([1]), and functions with FMI interfaces were investigated.

A) 1D simulator Linearizes the configuration of the element model defined using ordinary and algebraic differential equations for the fundamental equations and solves them as linear algebraic equations. Mechanism analysis is also equivalent to this.

B) 3D simulator Discretizes a 3D structure and solves discrete equations using finite element method, finite difference method, voxel method, etc. based on continuum mechanics. When it couples mechanism analysis and elastic body calculation and mainly uses them, it is classified as a 3D simulator.

C) Real-time simulator The simulator shall be operated as an emulator of plant models and an implementation of control algorithms on SILS and HILS, with execution in a real-time environment in mind.

The FMI homepage[(2)] provides information on various tools with FMI interfaces. The following is an excerpt regarding tool types and application fields. Table 2.2.1 shows a selection of tool types and fields of application. Note that this table excludes system simulation tools, tools with only mechanism analysis, and tools whose functions could not be verified.

Although multi-body simulators are the most commonly used and have many examples, we can also confirm that the use of these tools in the fields of fluid dynamics (CFD) and structural analysis (Structure) is also growing. PIDO (Process Integration and Design Optimization) in the table refers to process integration tools that focus on optimization and statistical analysis.

Table 2.2.1 Type and number of FMI-compliant CAE tools (2000)

Type	Application (Number)
(a)	Multi Body (6)
(b)	CFD (3)
(b)	Structure (2)
(c)	HILS (3)
Other	PIDO (3)

Based on this and other surveys, we investigated the use of tool linkage, and found that it is generallyFigure 2.2.1 Figure 2.2.1 shows the four use cases.

① Link 3D CAE tools with 1D system simulation as a Co-Simulation (CS) slave. Use solvers of 3D tools using communication, etc.

② Import and link 1D systems as boundary conditions to 3D tools.

③ The characteristic information of 3D models derived using optimization and process integration tools is condensed and operated in 1D systems and real-time simulators.

④ Collaboration to operate control and system simulation models in a real-time environment.

The 3D - 3D linkage has not been confirmed within the scope of the survey.

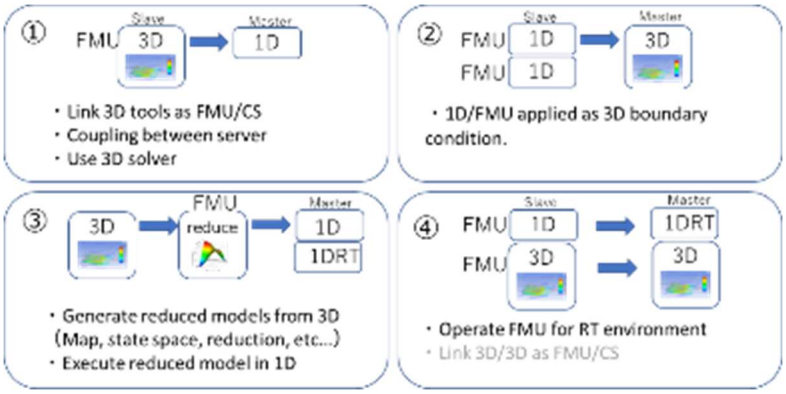
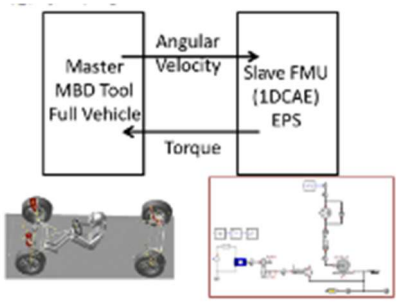


Figure 2.2.1 Types of use of tool linkage using FMI

2.2.3. Actual condition of 3DCAE collaboration

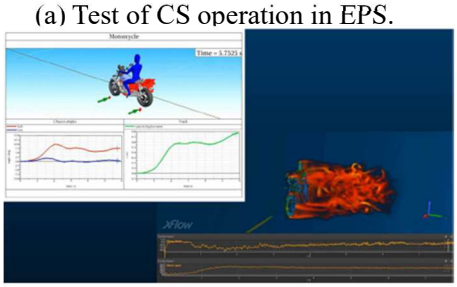
Application Examples are shown in Figure 2.2.2 through Figure 2.2.5. Case ① reports the testing of CS operation in EPS development[(3)] and the connection between crosswind analysis and recovery control logic using CFD[(4)] . In the former, the main deputy of CS and the connection through the CS control tool are used to verify the impact on execution time and the error accumulation. While the advantage is that it facilitates the realization of co-simulation, issues such as the formulation of communication step size are reported.

In case ②, the report includes vehicle behavior calculations and sloshing analysis in fuel tanks based on mechanism analysis and the effect analysis on vehicle behavior[(5)] , and the case[(6)] , which was used to develop control logic for equipment that responds to changes in stiffness during continuous forging. Some participants expressed their expectations for the use of FMU in this case because it allows 3D tool users to use advanced control and motion models in a tool environment with which they are familiar.



3D MultiBody ↔ 1D CAE

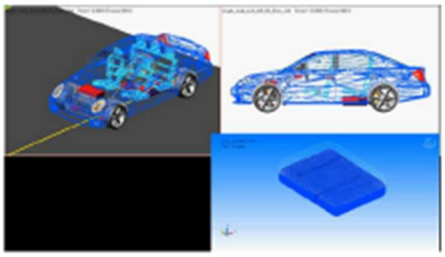
(a) Test of CS operation in EPS.



CFD → Multi Body

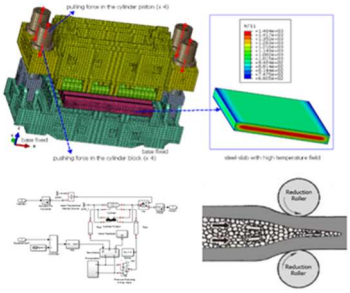
(b) Crosswind to MC and recovery control.

Figure 2.2.2 3D (slave) with 1D simulation.



Multibody→(CS)→ CFD

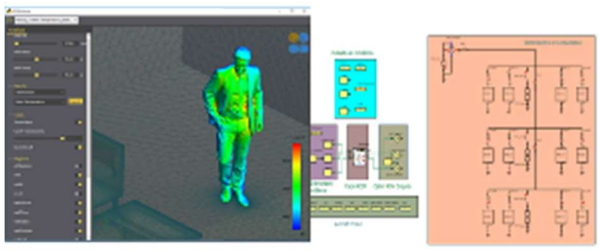
(a) Fuel sloshing on rough roads.



0D Sim → Structure

(b) Control for continuous forging.

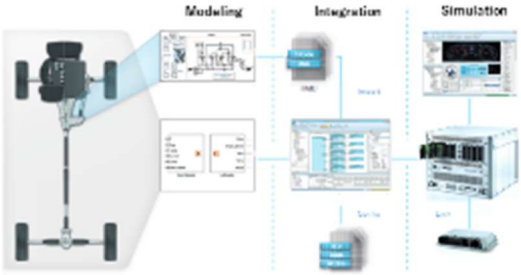
Figure 2.2.3 3D (master) with 1D simulation.



CFD →(ME)→ 1D Sim

HVAC using SVD reduced model[7].

Figure2.2.4 3D(reduced ME)with 1D simulation.



1D Sim →(CS)→ HIL

Verification of 8-speed AT-ECU.

Figure 2.2.5 1D with HIL simulation.

In case ③ (Figure 2.2.4) and case ④ (Figure 2.2.5), the scene of their utilization is also widespread. The reduced model can utilize the characteristics of an appropriate 3D model in a 1D simulation at high speed, and is expected to improve performance in the future. In addition, the environment for real-time execution is becoming better prepared, and in recent years, the number of compatible tools and hardware has been expanding, moving toward more realistic operations such as model parameter verification. The impact of these changes is an issue for further study.

2.2.4. Benefits and Challenges of 3D CAE Collaboration

The standardized interface of FMI enables advanced control and motion models to be applied as boundary conditions, and also minimizes the time required to acquire new tool operations. The standardized interface of FMI enables advanced control and the application of kinematic models as

boundary conditions, while minimizing the time required to acquire new tool operations. This facilitates the linkage with detailed control design using highly nonlinear physical models such as fluids and plastic deformation, and is expected to expand the field of application of 3D simulation.

For 1D system simulation engineers, it is also expected to be used for detailed 3D verification in the design phase and to front-load the design. Furthermore, the consistent use of models in SIL, MIL, and HIL will lead to the use of models as a solution for decision making in IoT platforms.

On the other hand, the scale of 3D simulation models tends to expand every year, and 3D computation time is undeniably a bottleneck due to its high cost. It is essential to take measures to speed up the process, such as parallelization and the use of GPUs, and future technological developments are expected. In addition, the FMI 2.0/CS standard treats support for Roll Back (time rewinding) as an option, which may cause problems when[(8)(9)] and multiple tools are used together. However, whether or not CFD for continuum calculation supports this function depends on the tool, and there are still issues to be solved when interfacing with complex mechanisms.

Although we were unable to confirm the effect of large-scale 3D tool collaboration in this survey, we would like to study in future surveys the examples of collaboration of multiple 3D tools (multiphysics, model merging, etc.) and issues such as overhead due to the amount of data communication when the model size is large.

References for this section

(1) MBD Study Group, Ministry of Economy, Trade and Industry: "Plant Model I/F Guidelines for Automobile Development Explanatory Document Ver. 1.0" , https://warp.da.ndl.go.jp/info:ndljp/pid/10341576/www.meti.go.jp/press/2016/03/20170331010/20170331010.html

(2) https://fmi-standard.org/tools/

(3) Hirono, Tanaka, and Matsumoto, "FMI coupling of multi-body analysis tools and 1DCAE tools in EPS development," Proceedings of the Society of Automotive Engineers of Japan 2016 Spring Meeting, pp2167-2170, (2016)

(4) Dssault Systems Tutorial

(5) https://www.otsuka-shokai.co.jp/event/region/19/0524cae/

(6) Application of FMI in Metal Casting press-Forming Process, Science in the Age of Experience 2016 paper, CISDI R&D Co.

(7) ANSYS 2019R1 Update Seminar

(8) Fabio, Marten, et al. "Step Revision in Hybrid Co-simulation with FMI", ACM/IEEE International Conference MEMOCODE 2016

(9) Ming, Ulas, et al. "Functional Mock-Up Interface Based Parallel Multistep Approach With Signal Correction for Electromagnetic Transients Simulations", IEEE Trans. Vo. 34, No. 3, May 2019

2.3. Coupling of C/C++, Python models with commercial simulation tools

2.3.1. use case

As described in the previous chapters, FMI has been adopted as one of the interfaces for coupled simulations in many commercial simulation tools. If the model to be coupled was developed using a commercial or open-source simulation tool that supports FMI, you can consider coupling with FMI using the methods introduced in the other chapters. This section introduces the coupling method when the model to be coupled is built in C/C++ language or Python.

The following two use cases will be discussed

Use Case 1: Coupling an in-house simulator with a commercial simulation tool
Use Case 2: Coupling machine learning and deep learning models with commercial simulation tools

Use case 1 is a case where an in-house simulator or simulation model developed in C/C++ is coupled with a commercial simulation tool that supports FMI. Since the interface of FMI is specified as a function of the C language, any programming language that has an FFI (Foreign Function Interface) to the C language can be coupled with FMU in principle. However, we will not go into the details here, but rather consider a case in which an in-house simulator developed in C/C++ is coupled with a commercial simulation tool.

Use Case 2 is a case where machine learning and deep learning models developed in the Python language are coupled with commercial simulation tools that support FMI. In recent years, models created using open-source libraries for machine and deep learning models, such as TensorFlow (https://www.tensorflow.org) and PyTorch (https://pytorch.org/), have been used in various aspects of engineering used in various aspects of activities. We will consider use cases where such models built in Python are coupled with commercial simulation tools.

2.3.2. How to realize Use Case 1

There are two ways to achieve Use Case 1

Type A: Implement FMI on the in-house simulator side and import it as a tool-coupled co-simulation FMU to a commercial simulation tool.
Type B: Add an FMU import function to the in-house simulator and import FMUs exported from commercial simulation tools into the in-house simulator.

Architecture for each of them, Figure 2.3.1 and Figure 2.3.2.

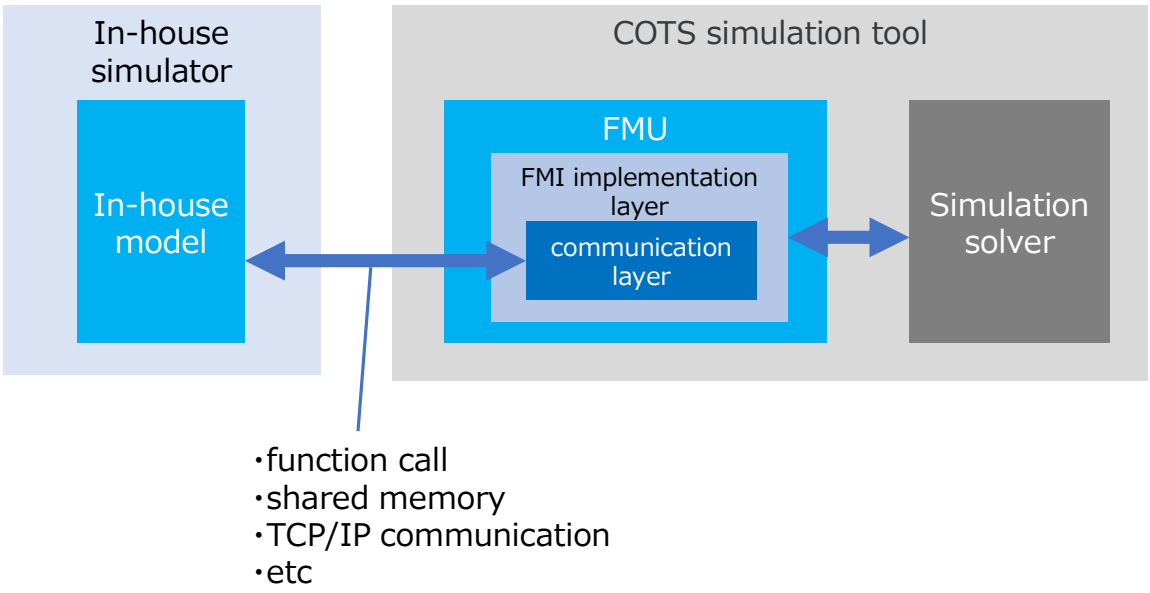


Figure2.3.1 When implementing FMI on the in-house simulator side

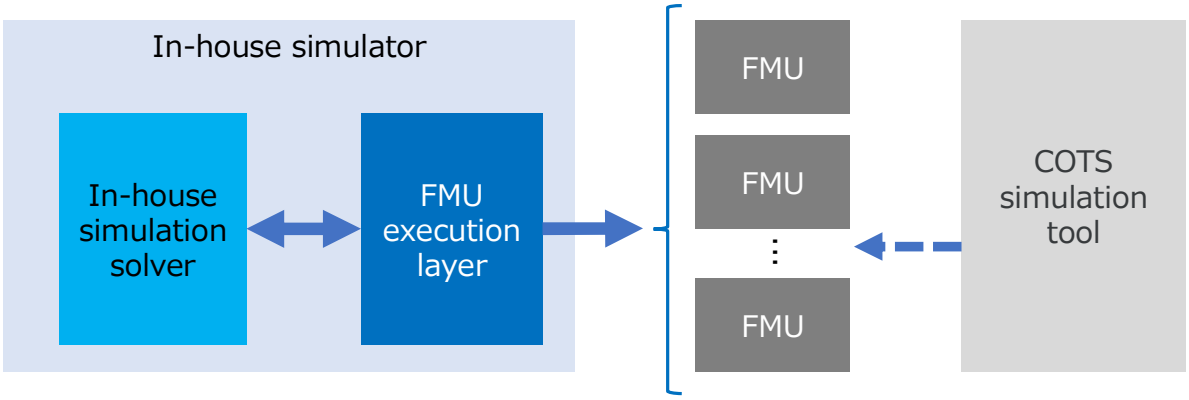


Figure 2.3.2 When implementing FMU import function on the in-house simulator side

In the case of type A, the FMI implementation layer and communication layer in Figure 2.3-1 must be implemented in order to be imported as an FMU into a commercial simulation tool. The FMI implementation layer is a layer that implements the C language functions defined in the FMI specification; although not many functions are required as FMI, the following open source software can be used for efficient development.

Name: Reference-FMUs
URL: https://github.com/modelica/Reference-FMUs
License: 2-license BSD license

Also, The communication layer in Figure 2.3.1 is the layer that realizes the specification by calling the in-house simulator when each function of the FMI is invoked. This can be achieved by means of function calls, inter-process communication using shared memory, etc., or TCP/IP communication. When using TCP/IP communication, for example, the following libraries can be used to achieve high-

speed communication.

Name: gRPC
URL: https://github.com/grpc/grpc
License: Apache License 2.0

To achieve Type B, an FMU execution layer must be implemented. In this case, there is no readily available open source software, but it is possible to implement it with a good understanding of the FMI execution model. and Co-Simulation) describes the execution model. Reference-FMUs, introduced earlier, includes a sample project, fmusim, for importing and executing FMUs. These reference files will help you understand what is required to implement the FMU import functionality.

2.3.3. How to realize Use Case 2

There are two ways to achieve Use Case 2

Type C: Implement an FMU that incorporates a Python processor and in-house Python program, and import it into a commercial simulation tool.
Type D: Add FMU execution capability using FMPy to the Python environment and import FMUs exported from commercial simulation tools into the Python environment.

Architecture for each of them, Figure 2.3.3 and Figure 2.3.4.

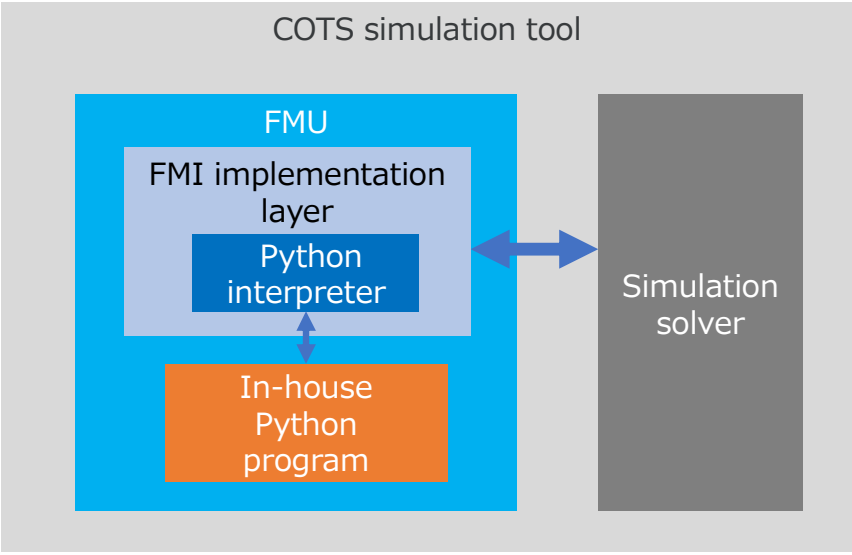


Figure2.3.3 Implementing a home-made FMU with an embedded Python processor

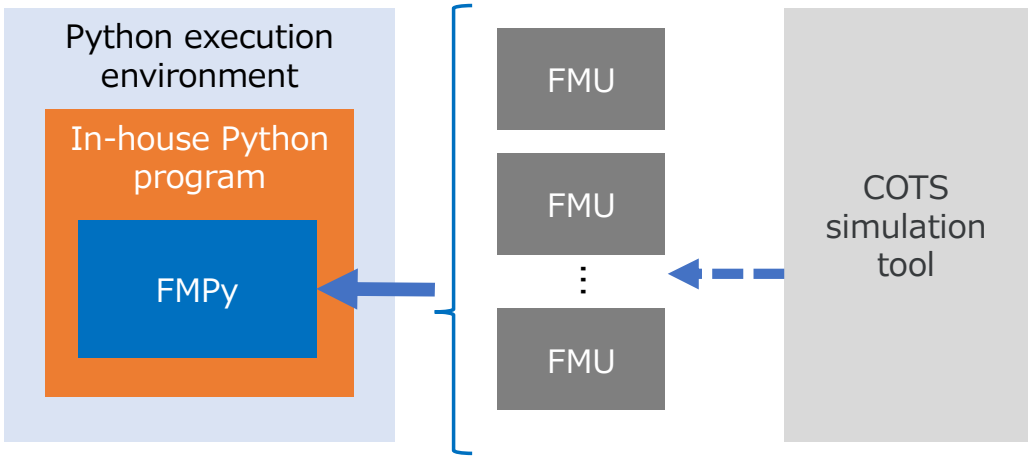


Figure 2.3.4 Importing an FMU into a Python environment

Type C is the same as Type A in that the FMI implementation layer is implemented using Reference FMUs, etc. However, Type C incorporates the Python processor into the FMU, since the Python processor has the ability to be embedded in other applications. The Python processor has the ability to be embedded in other applications, so this can be used to embed the entire Python execution function into FMU. This enables the execution of in-house Python programs when FMU functions are called, and allows commercial simulation tools to run Python programs as FMUs.

However, as a precaution, if your in-house Python program depends on a number of external libraries, such as machine learning and deep learning libraries, they must also be included in the FMU. All dependent libraries must be included in FMU
and that they work properly when imported into a commercial simulation tool as an FMU, it is often difficult to realistically employ Python programs of a certain scale in an FMU.

Type D is a way to add FMU execution capability to your own Python programs using FMPy, which is also described in Section 1.12 of this document.

Name: FMPy
URL: https://github.com/CATIA-Systems/FMPy
License: 2-license BSD license

FMPy is,1.12 section, but only the FMU execution functionality can be easily incorporated into an existing Python program. This makes it possible to implement a program in Python that runs a combination of your own Python program and FMU. Since the Python program does not need to be an FMU, it is possible to develop a program that works with an FMU even if it is a machine learning or deep learning model itself. Since this is easier to implement than Type C, we recommend using Type D if you want to couple a Python program with an FMU.

2.4. FMI-compliant in-vehicle communication protocol (CAN)

2.4.1. Challenges Facing Evolving E/E Architecture Development

As software creates new value for vehicles, user application software is becoming larger and more complex, and SW's efficient development methodology has become an essential requirement for advanced vehicle development.

With the evolution of E/E architecture, more and more devices tend to be installed, and SW verification with SWs installed in these devices interconnected with each other is essential.

If each device is available and can be interconnected, interconnected SW verification can be performed, but issues cannot be detected until the final stages of development, resulting in significant rework.

In addition, a device stand-alone simulation environment can be used to start SW development at an early stage. However, they cannot be interconnected, and system-level verification with multiple devices working together is not possible. This makes it difficult to ensure quality because verification is based on complex scenarios that are assumed.

2.4.2. Overview of Multi-Device Co-simulation Environment

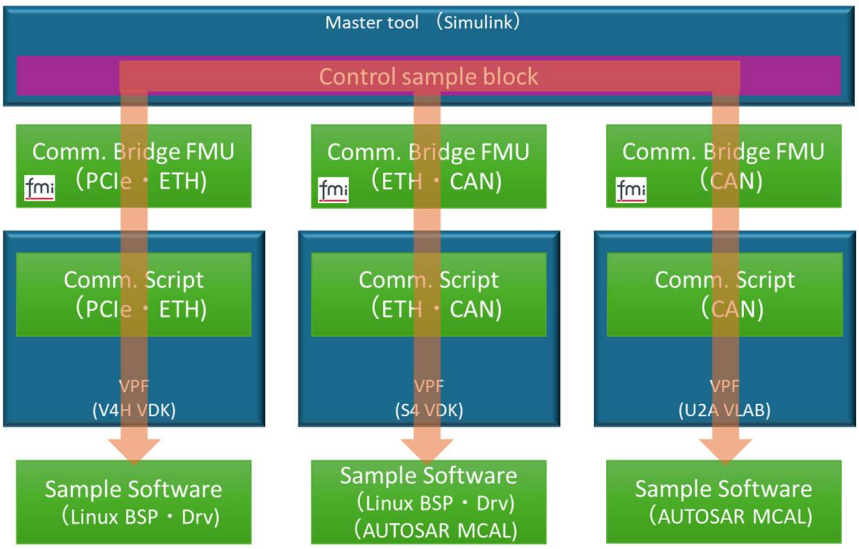


Figure2.4.1 Configuration of a multi-device co-simulation environment

・Overview of Multi-Device Co-Simulation Environment

[Configure the **Co-simulation** Environment for **Multi-device Co-simulation]** in Figure 2.4.1.

Renesas proposes a multi-device co-simulation environment that enables devices to be interconnected and software development to begin without the need for actual devices.

- FMU (*2) and control sample block for easy connection of each VPF via master tool (*1)
- Communication script to connect FMU and VPF
- VPF(*3), which simulates the operation of each actual machine
- Communication sample SW operating on interconnecting VPF

By using the "Software Development Kit", interconnected software development can be started before the actual equipment is available.

Since software developed on VPF also runs on the actual device, software development and verification on the actual device can be conducted without delay after obtaining the actual device. This enables early development of interconnected SWs and early detection of problems. (Master tool and VPF must be obtained by the user.)

(*1) Master Tool A tool to adjust and synchronize the operation of each VPF
(*2): Function Mockup Unit (FMU) A library for connecting tools according to the Function Mockup Interface (FMI), a model IF for connecting different tools. By importing from the master tool, interconnection is possible within the master tool.
(*3): Virtual Platform (VPF) A simulator for SOCs and MCUs that can execute SWs similar to real devices.

2.4.3. Measures for connection between CAN models

[How to realize CAN communication between models connected by FMI]Figure 2.4.2). The FMI defines what type and when data can be sent and received, but does not define specific connection specifications. It is left to the user to determine which parameters and how to connect, and it is necessary to define the connection specifications in the FMI.

For this reason, in a multi-device co-simulation environment, Figure 2.4.2 shows a CAN connection specification for connecting FMI import blocks. Models with FMIs that conform to this specification can be connected to each other, even between different tools.

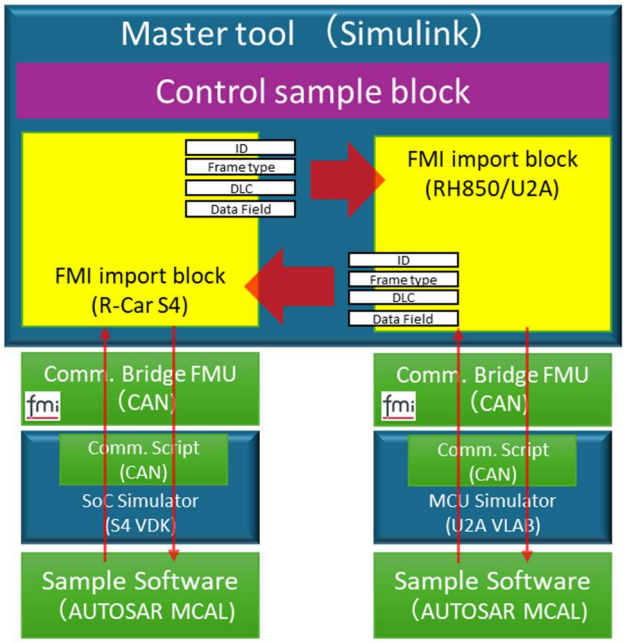


Figure 2.4.2 How CAN communication is realized between models connected by FMI

2.4.4. Development status of cooperative simulation environment for multi-device

In this presentation, we introduced the cooperative simulation environment for multi-device with R-CarS4 and RH850/U2A connected via CAN.

(Introduction of Virtual Platform Co-simulation | Renesas) for a demo video recording of actual operation.

We also verified a multi-device cooperative simulation environment in which R-CarS4 and R-CarV4H are connected via Ethernet, and an environment in which models simulating CAN and Ethernet buses are prepared on Simulink.

Thus, the commonization (publication) of communication protocol specifications by FMI facilitates the connection between different tools, and the realization of early SW development in a large-scale simulation environment offers the prospect of contributing to the development of SW-First, which determines the overall system configuration from the SW structure.

Chapter 3 Tutorial

In this chapter, we have provided practical examples for utilizing FMI.

3.1. Tutorial 1: Let's try using FMI

3.1.1. Let's prepare a sample model.

Let's actually take the sample model provided on the Society of Automotive Engineers of America website's Automotive Controls and Models Section Committee page and run the simulation. The models will be benchmark models from a presentation given by our WG at the Society of Automotive Engineers of America Spring Meeting in May 2018. [1]

File name：Sample_3p1.zip ：ZIP file contains the following four files.

FMU1 (Model Exchange) ：FMU_J1_Case1_Dymola_ME_2.fmu

FMU2 (Model Exchange) ：FMU_SD1_Case1_Dymola_ME_2.fmu

FMU1 (Co-Simulation) ：FMU_J1_Case1_Dymola_CS_2.fmu

FMU2 (Co-Simulation) ：FMU_SD1_Case1_Dymola_CS_2.fmu

This example is a simple rotary spring damper model that has been split, and the FMU has been created by

Version ：FMI 2.0

Type ：ME and CS

Compiler ：Visual C++ 64bit

OS ：Windows 64bit

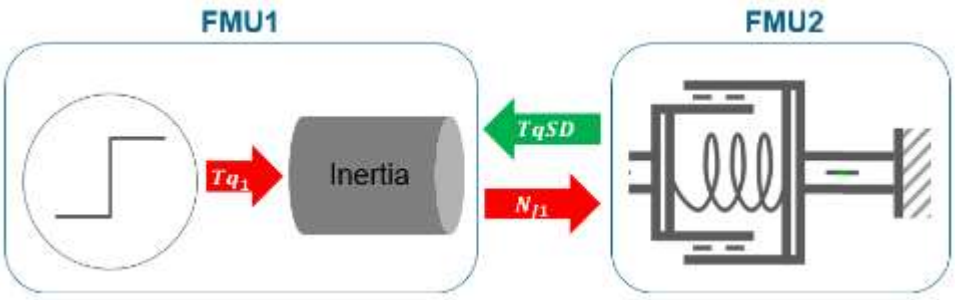


Figure 3.1 .1 Model Structure

First, you need to prepare a simulation tool that supports Import. Please check the FMU Import compatibility status of your tool at the FMI website described in section 2.9 of this volume.

Once the tools are available, the first step is to import the FMU, but the procedure varies depending on the tool used.

Did you get it right?

The following is an example of execution in Simcenter Amesim.

3.1.2. Model Exchange Simulation

3.1.2.1. Run with standard parameters

Next, we will connect the imported FMUs. First, both FMU1 and FMU2 use Model Exchange for both FMU1 and FMU2. Since each FMU has only one input and one output, connecting the outputs and inputs of both FMUs should be sufficient(Figure 3.1.2).

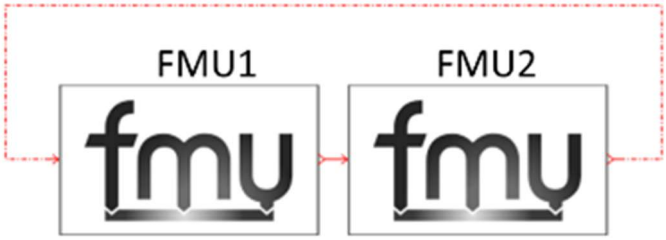


Figure 3.1 .2 FMU Connection

Next, set the FMU parameters.

Figure3.1.3 a and Figure3.1.3b's parameter "path to the unzipped FMU root" (depending on the tool) in specifies the location where the unzipped FMU dll of the unzipped FMU. The path is important because the simulation will call the dll at this address.

The value beginning with # is the initial value of the state variable for the integral. Although it can be changed, we will start with this setting. Note that variables such as # are displayed differently depending on the tool used.

Parameters

Title	Value	Unit
(#) inertia.phi - Absolute rotation angle of component	0	rad
(#) inertia.w - Absolute angular velocity of component (= der(phi))	0	rad/s
step.height - Height of step	1	null
step.offset - Offset of output signal y	0	null
step.startTime - Output y = offset for time < startTime	1	s
inertia.J - Moment of inertia	1	kg.m2
TqSD - Accelerating torque acting at flange (= -flange.tau) - start value	0	N.m
▾ Import parameters		
enable logging	no	
path to the unzipped FMU root	.../FMU_J1_Case1_Dymola_ME_J1	

Figure3.1.3 a FMU1 (Model Exchange) Parameters

Parameters

Title	Value	Unit
(#) speed.phi - Rotation angle of flange with respect to support	0	rad
spring.c - Spring constant	1	N.m/rad
spring.phi_rel0 - Unstretched spring angle	0	rad
damper.d - Damping constant	1	N.m.s/rad
fixed.phi0 - Fixed offset angle of housing	0	rad
speed.f_crit - if exact=false, critical frequency of filter to filter input s...	50	Hz
Nj1 - Reference angular velocity of flange with respect to support as i...	0	rad/s
▾ Import parameters		
enable logging	no	
path to the unzipped FMU root	.../FMU_SD1_Case1_Dymola_ME_SD1	

Figure3.1.3 b FMU2 (Model Exchange) Parameters

Let's quickly run the simulation with the Default parameter set in the FMU to see the results. The rotation speed Nj1 and torque TqSD are as shown in Figure 3.1.4, right?

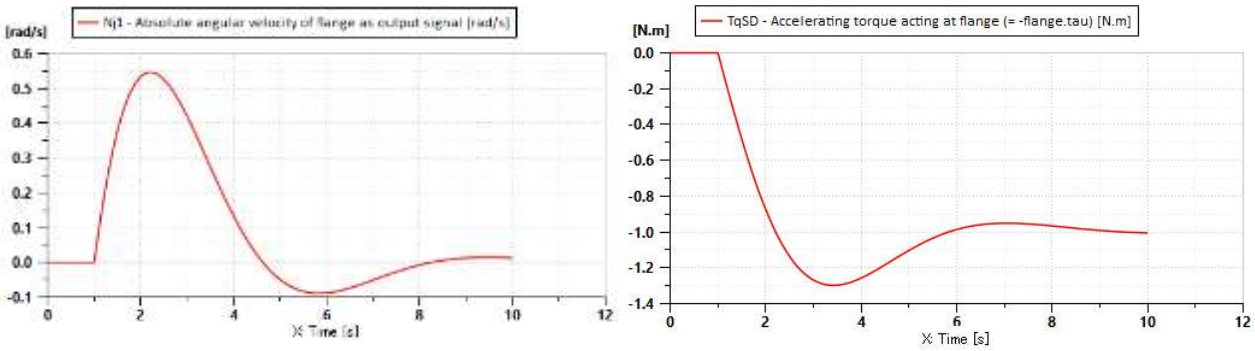


Figure 3.1.4 Simulation Results (Model Exchange)

3.1.2.2. Change parameters and execute

Next, let's change the parameters. Increase the resonance frequency and set the damping to a lower value.

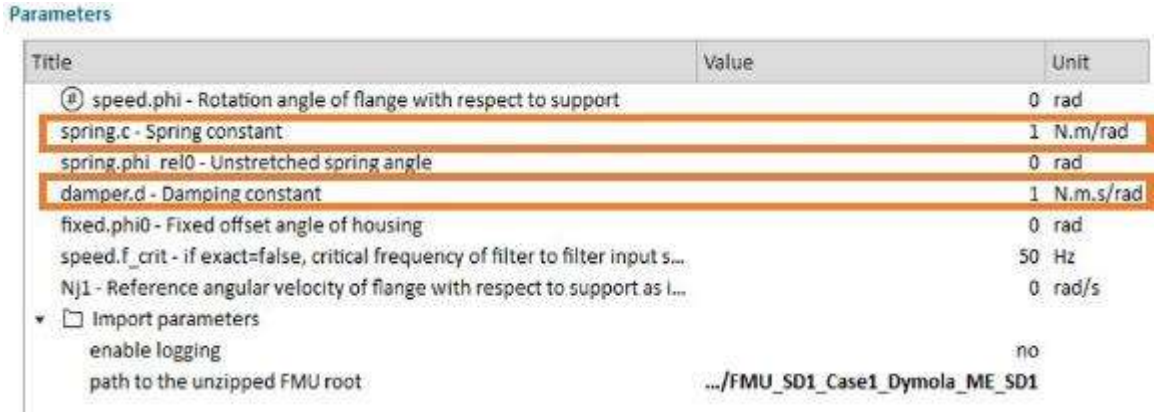


Figure 3.1.5 FMU1 (Model Exchange) Parameters before modification

- Spring constant (spring.c-Spring constant):
  Changed from the current "1 [N-m/rad]" to "10 [N-m/rad]" (10 times)
- Damping (damper.d-Damping constant):
  Changed from the current "1 [N-m-s/rad]" to "0.1 [N-m-s/rad]" (1/10)

Run the simulation after the change and check the results.

The rotation speed Nj1 and torque TqSD are as shown in Figure 3.1.6?

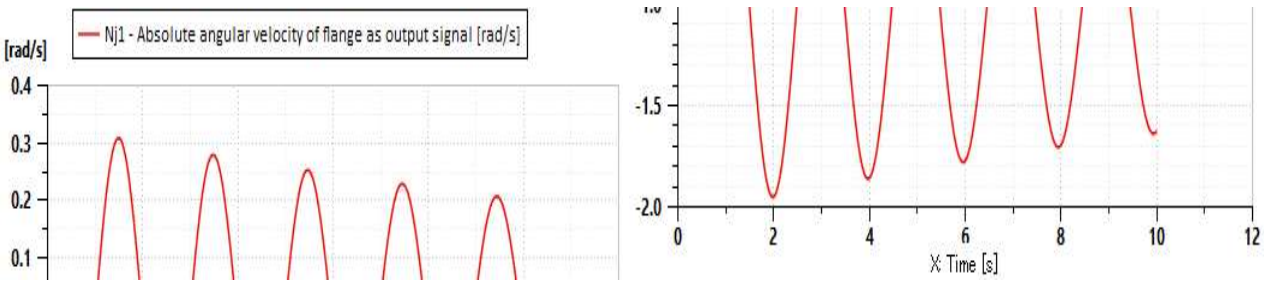


Figure 3.1.6 Simulation results (ME: after parameter changes)

3.1.3. Co-Simulation Let's simulate

3.1.3.1. Run with standard parameters

Next, let's import the Co-Simulation FMU, connect the model, and check the parameters.

The parameters are almost the same as in Model Exchange The parameters are almost the same as in Model Exchange, but the following two parameters have been increased for each FMU.

- co-simulation step size
- co-simulation step specification

Parameters

Title	Value	Unit
inertia.phi - Absolute rotation angle of component	0	rad
inertia.w - Absolute angular velocity of component (= der(phi))	0	rad/s
step.height - Height of step	1	null
step.offset - Offset of output signal y	0	null
step.startTime - Output y = offset for time < startTime	1	s
inertia.J - Moment of inertia	1	kg.m2
TqSD - Accelerating torque acting at flange (= -flange.tau) - start value	0	N.m
Import parameters		
co-simulation step size	0.001	s
co-simulation step specification	time step size	
enable logging	no	
path to the unzipped FMU root	.../FMU_J1_Case1_Dymola_CS_J1	

Figure 3.1.7 a FMU1 (Co-Simulation) Parameters

Parameters

Title	Value	Unit
speed.phi - Rotation angle of flange with respect to support	0	rad
spring.c - Spring constant	1	N.m/rad
spring.phi_rel0 - Unstretched spring angle	0	rad
damper.d - Damping constant	1	N.m.s/rad
fixed.phi0 - Fixed offset angle of housing	0	rad
speed.f_crit - if exact=false, critical frequency of filter to filter input s...	50	Hz
Nj1 - Reference angular velocity of flange with respect to support as i...	0	rad/s
Import parameters		
co-simulation step size	0.001	s
co-simulation step specification	time step size	
enable logging	no	
path to the unzipped FMU root	.../FMU_SD1_Case1_Dymola_CS_SD1	

Figure 3.1.7 b FMU2 (Co-Simulation) Parameters

These are parameters that set the communication interval for Co-Simulation These are the parameters that set the communication interval when performing Co-Simulation. The names of the parameters here may vary depending on the tool used. In this case, "0.001 [s]" is set as the communication interval, and inputs to each FMU are exchanged at this interval.

Now let's run the simulation with the Default parameter set set in the FMU and check the results.

The rotation speed Nj1 and torque TqSD are as shown in Figure 3.1.8?

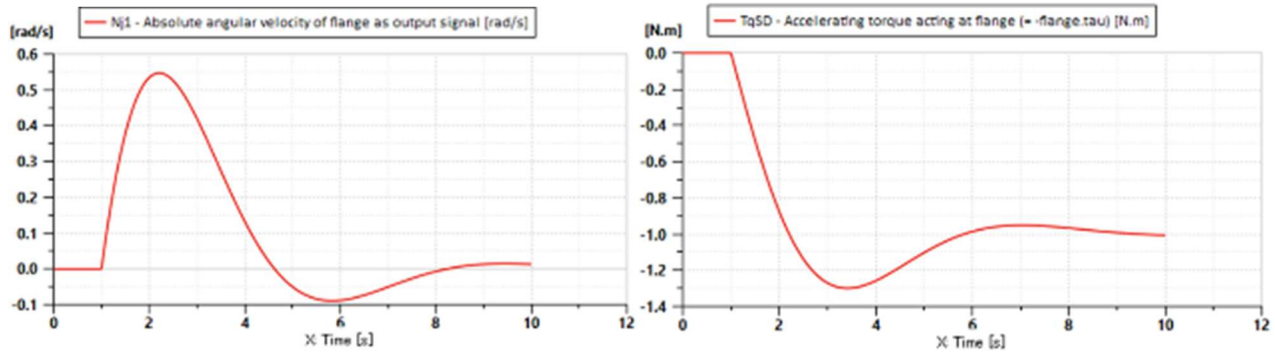


Figure 3.1.8 Simulation Results (Co-Simulation)

Model Exchange Similar simulation results are seen. However, when performing Co-Simulation is performed, the input/output of the FMU is treated as a constant value between communication intervals, and a Step Delay (see Section 2.4.2 of this volume). In this example, the communication interval is as fine as 0.001[s], which is difficult to recognize. Let's try to expand the graph by using a finer sampling setting for the output results (e.g., 0.0001[s]).Figure 3.1.9 shows that the result changes discretely every 0.001[s] of the communication interval.

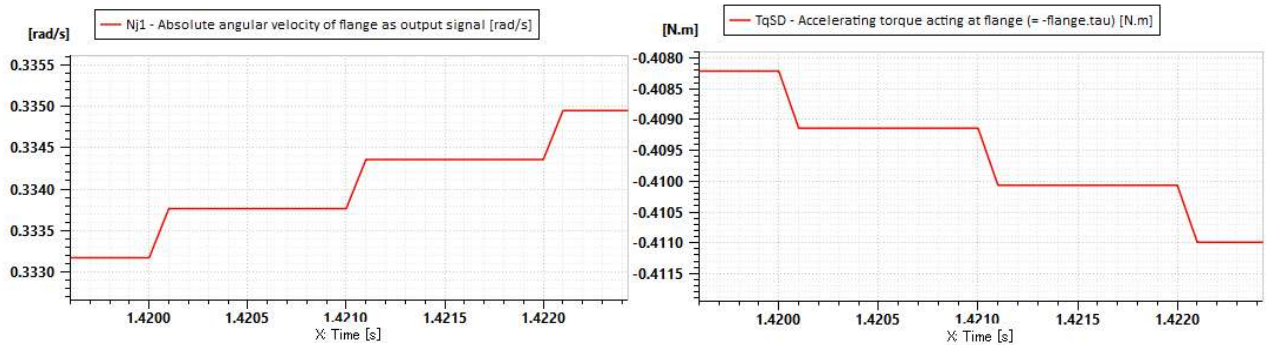


Figure 3.1.9 Simulation Results (Enlarged Figure 3.1.9)

3.1.3.2. Change parameters and execute

Next, let's change the parameters: coarsely set the communication interval for FMU1 and FMU2.

- Communication interval (co-simulation step size): Changed from the current "0.001 [s]" to 100 times "0.1 [s]".

The communication interval is coarser, and discrete results are more easily visible. Also, when compared to the results with a communication interval of 0.001[s], it can be seen that the results are different. In the Co-Simulation, it is clear that setting the appropriate communication interval is important. Please refer to Chapter 4 for more details.

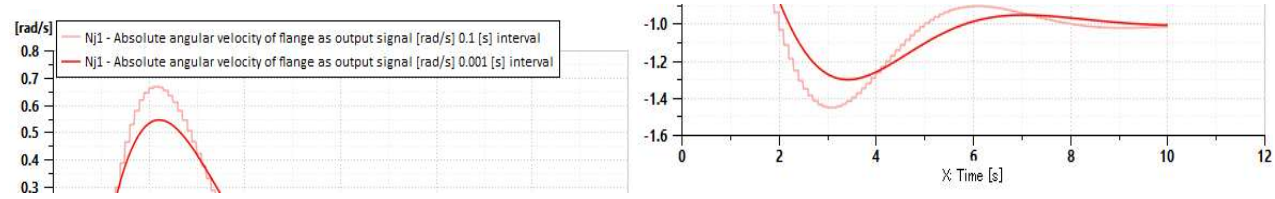


Figure 3.1.10 Simulation Results (CS: Comparison before and after parameter change)

3.2. Tutorial 2: Let's make an FMI

3.2.1. Let's generate an FMU using the sample model

Sample models are stored on the Society of Automotive Engineers of America website on the Automotive Controls and Models Section Committee page. The models are benchmark models from a talk given by our WG at the Society of Automotive Engineers of America Spring Conference in May 2018. [1]

File name：Sample_3p2.zip　　ZIP file contains the following files.

Table 3.2.1　Sample Model Contents

	Model Exchange Original Model		Co-Simulation Original Model	
Creation Tools	For FMU1	For FMU2	For FMU1	For FMU2
Amsim	J1_Amesim_ME.ame	SD1_Amesim_ME.ame	J1_Amesim_CS.ame	SD1_Amesim_CS.ame
Simplorer	J1_SD1_Simplorer_ME_CS.aedt (all 4 types in one model)			
Dymola	J1_Dymola_ME.mo	SD1_Dymola_ME.mo	J1_Dymola_CS.mo	SD1_Dymola_CS.mo
MapleSim	J1_MapleSim_ME.msim	SD1_MapleSim_ME.msim	J1_MapleSim_CS.msim	SD1_MapleSim_CS.msim
SimulationX	J1_SimulationX_ME.isx	SD1_SimulationX_ME.isx	J1_SimulationX_CS.isx	SD1_SimulationX_CS.isx
OpenModelica	J1_Modelica _ME.mo	SD1_Modelica _ME.mo	-	-

If you have a tool that corresponds to the above table 3.2.1, you can generate an FMU. Note that the procedure for generating FMUs varies depending on the tool used, so please refer to the manual of the tool for details.

Here is an example of generating an FMU using Simcenter Amesim.

First, open the Co-Simulation open the file J1_Amesim_CS.ame for FMU1, the original model of Co-Simulation. Configuration is like Figure 3.2.1, but this model already has the FMI interface block inserted and is ready to generate an FMU.

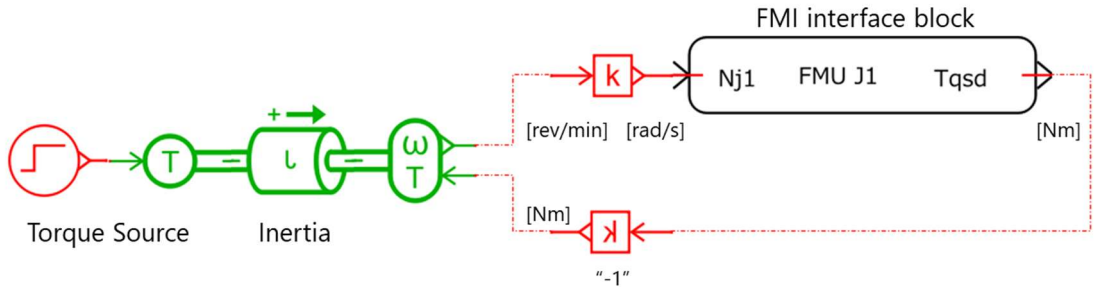


Figure 3.2.1　Configuration of J1_Amesim_CS.ame (J1 model) for FMU1 output

In generating FMUs, attention should be paid to the input/output unit system and the positive/negative direction of the signals.

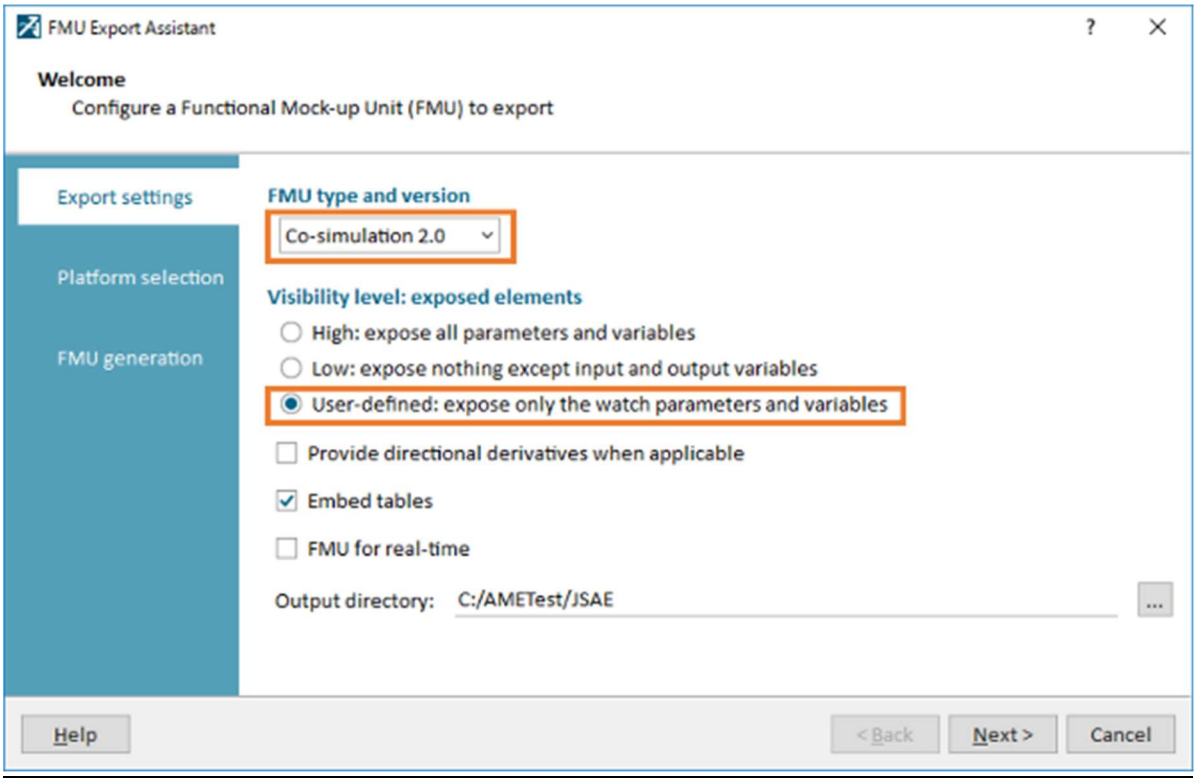
The rules for model exchange are based on METI guidelines (see Section 4.1.1 of this volume). Since the SI assembly unit system is used as the unit of exchange between models, [rad/s] is used for rotational speed and [Nm] for torque.

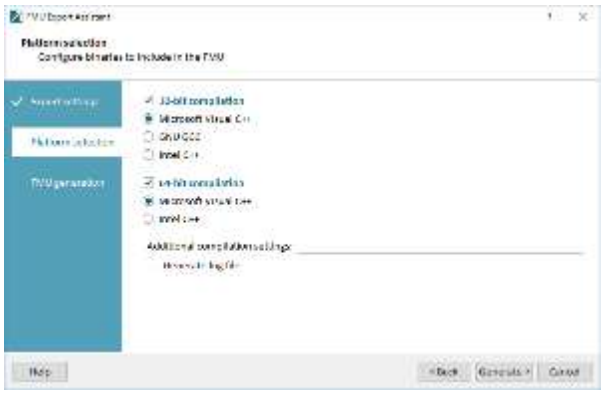
In Amesim, the RPM sensor outputs [rev/min], which is converted to a unit with "k" and outputs Nj1 to the FMU2 side.

For the positive and negative directions of the signal, the energy source to energy sink is the positive direction of energy. In this model, the flow from the torque source to the inertia to the spring + damper on the FMU2 side is in the positive direction. the torque coming from the FMU2 side works in the opposite direction of energy transfer, so when the inertia rotates in the positive direction, the torque returns in the negative direction to reduce the rotation speed. The torque coming in from the FMU2 side works in the opposite direction to the energy transfer.

Amesim's model defines this torque as positive, so a gain of -1 is applied with "k" to make the model invert the positive and negative directions. Note that the treatment of positive and negative physical quantities, not limited to torque in this example, varies depending on the tool and the way the model is created.

To export the model to FMU, launch Interface > FMU Export Assistant. Follow the menu of the wizard in Figure 3.2.2 to generate an FMU.





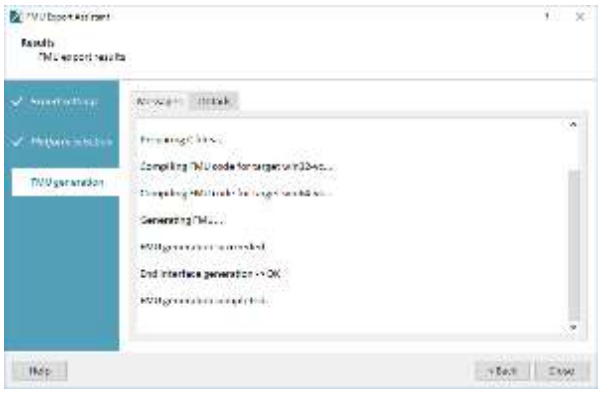


Figure 3.2.2　FMU Export Assistant

Select "FMU type and version" for FMI 2.0 co-simulation.

Next, set the "Visibility level: exposed elements" to determine the display level of the parameters, and select "User-defined: expose only the watch If "User-defined: expose only the watch parameters and variables" is selected, the parameters and variables to be displayed in the FMU can be set arbitrarily, making the model easy to use for the users after distribution. For a concrete example, see Section 3.3 .

Then select the compiler to be used to generate the FMU.

Once the FMU is generated, import it again and run the simulation.

3.2.2. Display only the parameters you need!

During FMU generation, depending on how the underlying model is built, unnecessary parameters may be displayed that do not affect the simulation results. The display of unnecessary parameters and variables can lead to confusion on the part of the FMU user.

In this section, Model Exchange used in the sample model in FMU2 used in the sample model in 3.1.2.Figure 3.2.3, the parameter "speed.f._crit - if exact=false, critical frequency of filter to filter input s..." in reality, changing this parameter has no effect on the simulation results. Why does this happen?

Parameters

Title	Value	Unit
speed.phi - Rotation angle of flange with respect to support	0	rad
spring.c - Spring constant	1	N.m/rad
spring.phi_rel0 - Unstretched spring angle	0	rad
damper.d - Damping constant	1	N.m.s/rad
fixed.phi0 - Fixed offset angle of housing	0	rad
speed.f_crit - if exact=false, critical frequency of filter to filter input signal	50	Hz
Nj1 - Reference angular velocity of flange with respect to support as input signal - start value	0	rad/s
Import parameters		
enable logging	no	
path to the unzipped FMU root	../FMU_SD1_Case1_Dymola_ME_SD1	

Figure 3.2.3 FMU2 (Model Exchange) Parameters

The original model before generating FMU2 was created with Modelica shows Figure 3.2.4. Open SD1_Dymola_ME.mo (SD1 model) if you have the corresponding tool.

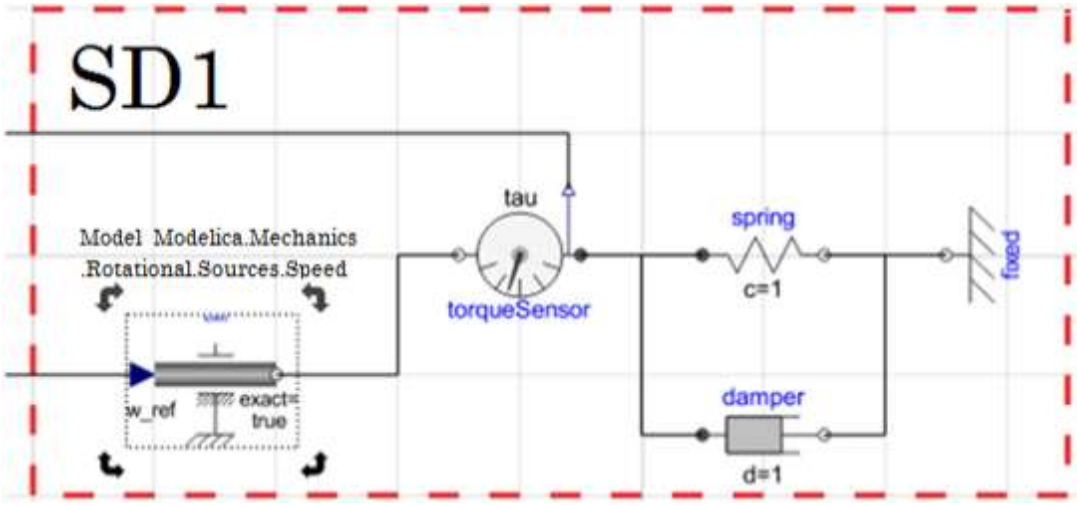


Figure 3.2.4 Original Modelica (SD1 model) configuration

This model uses the "Model Modelica.Mechanics. Sources.Speed" and the parameters of this block are Figure 3.2.5.

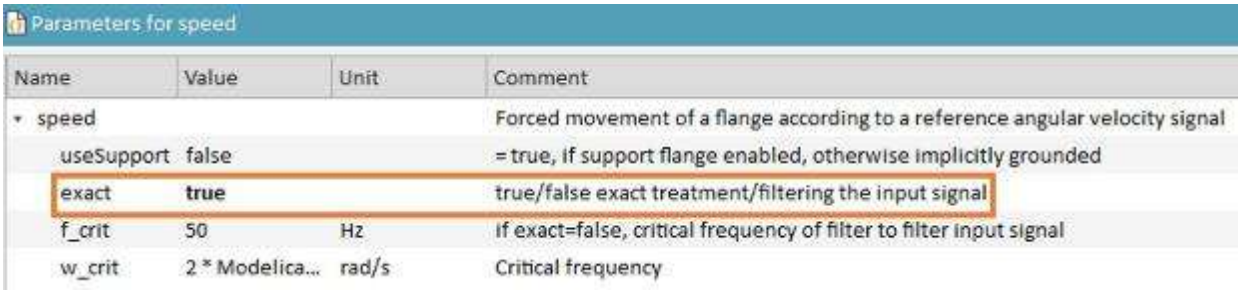


Figure 3.2.5 (SD1 model) Parameters of speed

There is a parameter "f_crit", but it is only enabled when "execute" is set to "false". If set to "true" at the time of FMU generation, it cannot be changed after the FMU is generated.

Thus, in Figure 3.2.5, if “speed.f._crit - if exact=false, critical frequency of filter to filter input s... “ will be invalid.

This phenomenon is caused by treating “f_crit” as a modifiable parameter when generating FMUs, so only necessary parameters and variables should be displayed, and parameters that do not need to be changed should be hidden. This will prevent confusion among users.

3.3. Tutorial 3: Example of our WG benchmark model

The Society of Automotive Engineers of Japan Open Committee "Model Connection Workshop by FMI (Functional Mockup Interface)" in October 2017 and May 2018 2nd Japanese Modelica Conference The tutorial will proceed with an example of our WG's benchmark model [2]. w

3.3.1. Description of Sample Models

This vehicle model consists of five FMUs for driver, powertrain, driveline, chassis, and steering for Co-Simulation Co-Simulation with five FMUs: driver, powertrain, driveline, chassis, and steering. The outline is shown inFigure 3.3.1 for an overview.

Please note that the OS is Windows 64bit version, FMI2.0 Please note that only Windows 64-bit version and FMI2.0 are supported.

Sample models are available on the Automotive Controls and Models Division Committee page of the Society of Automotive Engineers of Japan website of the Society of Automotive Engineers of Japan (JSAE) website.

File name：Sample_6p2.zip : ZIP file contains the following 7 files.

Driver	: Driver_FMU_Export_Rev16.fmu.fmu
Power Train	: Powertrain_FMU_export_Rev16.fmu
Drive Line	: Drive Line: DS_Simulink.fmu
Chassis	: FMI_20_Blackbox_Euler_Test_FlatPad.fmu
Steering	: SteeringMechanismRackPinionInMetric.fmu
Subsystem I/F Definition Sheet	: Sample_6p2_Subsystem_I/F_Definition.xlsx
Simulation Results	: Sample_6p2_Result.txt

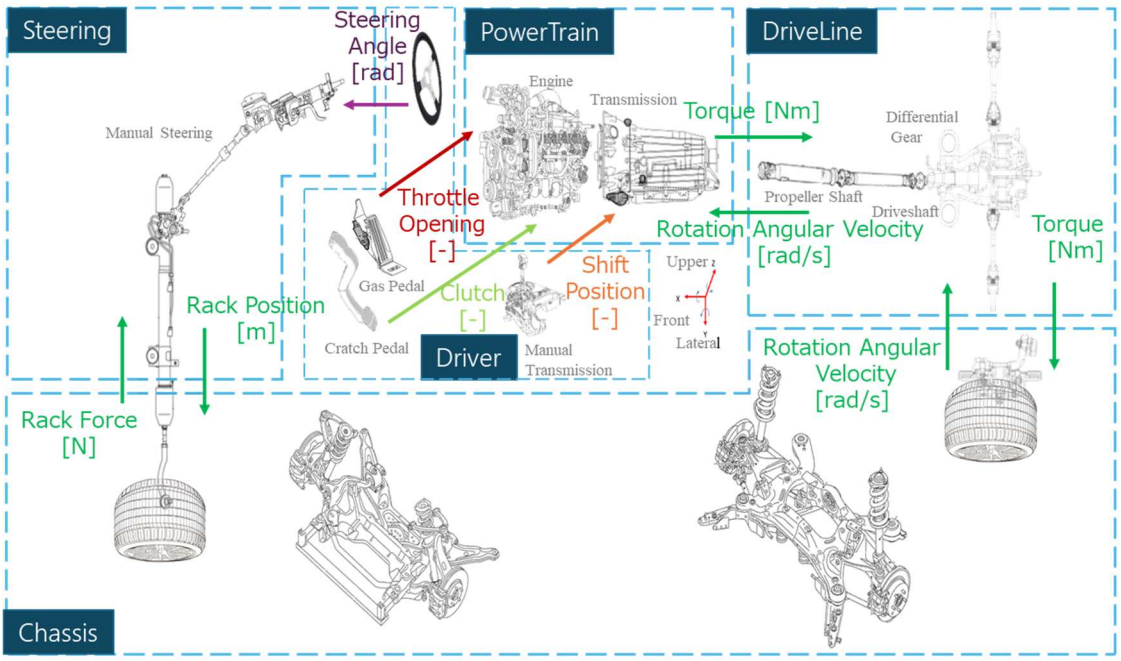


Figure 3.3.1 Overview of Benchmark Model

First, import the 5 FMUs into the simulation tool to be used. The method of importing depends on the tool, such as specifying on the menu screen or dragging and dropping from Explorer, etc. Please refer to the manual. If you want to know the contents of each FMU, please refer to the Subsystem I/F Definition Document (Sample_6p2_Subsystem I/F Definition.xlsx) included in the ZIP file.

3.3.2. Model Connections

Next, connect each FMU.Table 3.3.1 for details. The green hatched area is not used for connection because it is an output for monitoring. Figure 3.3.2 shows a model connection example. The left terminal of each FMU is an input and the right terminal is an output. Note that the appearance of the connection varies depending on the tool used.

Table 3.3.1 Input/output table

FMU Number	Terminal Number		Terminal Number	INPUT Signal			FMU Number	FMU Name	Output Signal			Terminal Number
							1	Driver	Clutch_Action@expseu_	[-]	Clutch	1
									Engine_load@expseu_	[-]	Throttle opening	2
									Gear_position@expseu_	[-]	Shift position	3
									SteerinAngle@expseu_	[rad]	Steering Angle	4
3	1	⇒	1	Npropshaft@expseu_	[rad/s]	Propeller shaft rotation speed	2	Power Train	Neng@expseu_	[rad/sec]	Engine side rotation speed	1
1	3	⇒	2	Gear_position@expseu_	[-]	Shift position			Nclutch@expseu_	[rad/sec]	Clutch side rotation speed	2
1	2	⇒	3	Engine_load@expseu_	[-]	Throttle opening			Tpropshaft@expseu_	[Nm]	Propeller shaft torque	3
1	1	⇒	4	Clutch_Action@expseu_	[-]	Clutch						
2	3	⇒	1	Tq_TM	[Nm]	Propeller shaft torque	3	Drive Line	N_Propshaft	[rad/sec]	Propeller shaft rotation speed	1
4	7	⇒	2	N_RL	[rad/s]	DriveShaft rotation speed RL			Tq_RL	[Nm]	Drive Shaft Torque RL	2
4	5	⇒	3	N_RR	[rad/s]	DriveShaft rotation speed RR			Tq_RR	[Nm]	Drive Shaft Torque RR	3
5	2	⇒	1	steeringTranslation	[m]	Rack Position	4	Chassis	position_x	[m]	Vehicle Transration X	1
Constant[0]		⇒	2	FrontRightWheel_Tau	[N]	Drive Shaft Torque FR			position_y	[m]	Vehicle Transration Y	2
3	3	⇒	3	RearRightWheel_Tau	[N]	Drive Shaft Torque RR			position_z	[m]	Vehicle Transration Z	3
Constant[0]		⇒	4	FrontLeftWheel_Tau	[N]	Drive Shaft Torque FL			FrontRightWheel W	[rad/sec]	DriveShaft rotation speed FR	4
3	2	⇒	5	RearLeftWheel_Tau	[N]	Drive Shaft Torque RL			RearRightWheel W	[rad/sec]	DriveShaft rotation speed RR	5
		⇒							FrontLeftWheel W	[rad/sec]	DriveShaft rotation speed FL	6
		⇒							RearLeftWheel W	[rad/sec]	DriveShaft rotation speed RL	7
		⇒							Steering_force	[N]	Rack Force	8
1	4	⇒	1	steerAngle	[rad]	Steering Angle	5	Steering	Steer_Torque	[Nm]	Steering Torque	1
4	8	⇒	2	rackForce	[N]	Rack Force			rackPosition	[m]	Rack Position	2

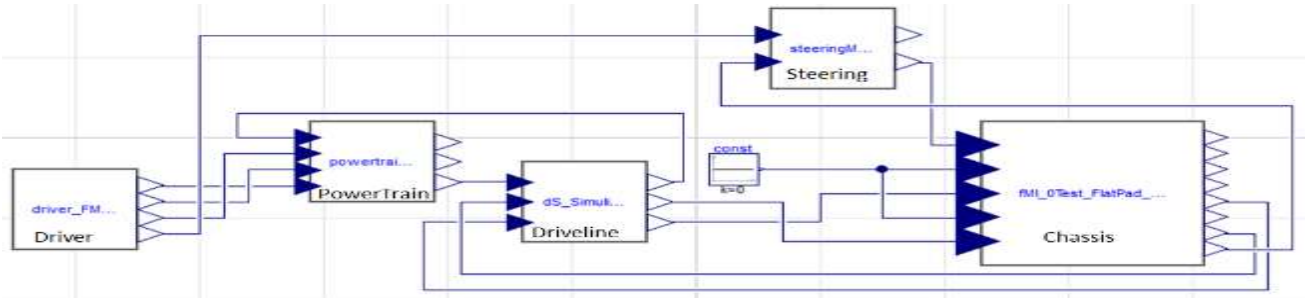


Figure3.3.2 Model Connection Example

3.3.3. Driver Model Description

The Open loop is a simple input operation as shown in Figure 3.3.3. The manual transmission specification requires clutch operation and sequential gear shifting from stop to 5th gear.

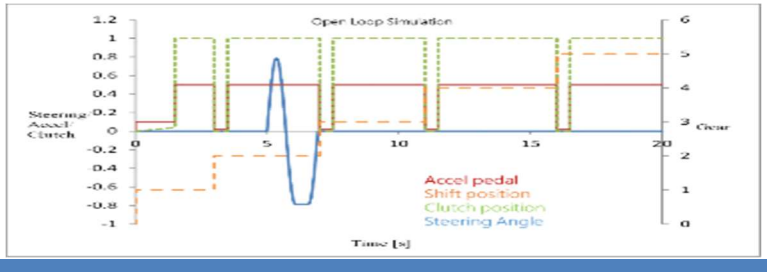


Figure 3.3.3 Driver Model Behavior

3.3.4. Setting FMU parameters

The default parameters cause the tires to cut to the right in response to forward (left-turn) steering operation, so the setting must be changed.Figure3.3.4 is an overview of the steering model. To reverse the direction of movement of "rackPosition" in response to the "steerAngle" input from the driver model, the parameters of the "gs1 block, indicated by the red frame, to reverse the direction of movement of the "rackPosition" for the "steerAngle" input from the driver model.

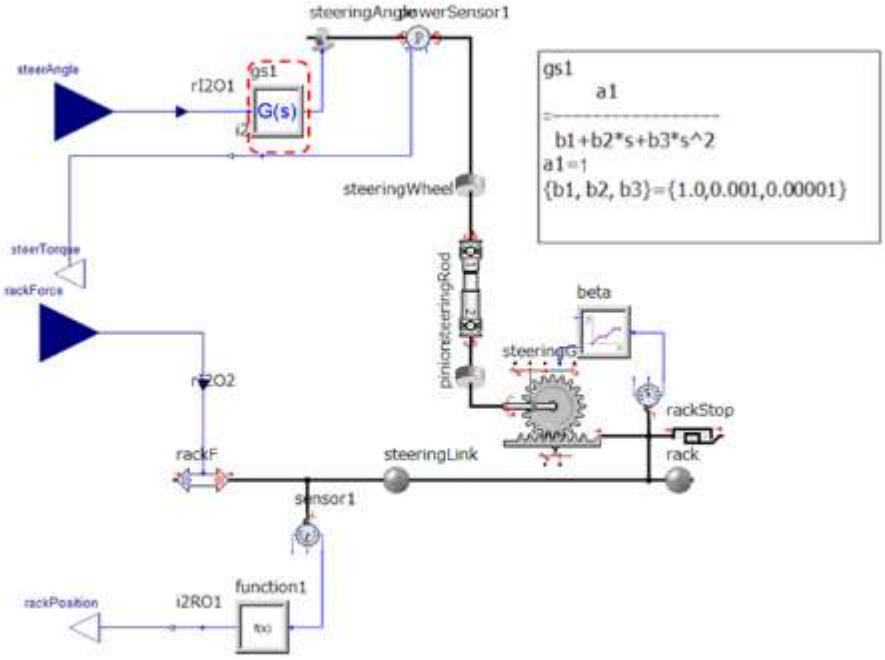


Figure3.3.4 Overview of the Steering Model

In the SimulationX example, The settings are made in the Parameters tab as shown in Figure 3.3.5.

- Numerator of quadratic transfer function (gs1.A[1]): changed from current "1 [-]" to "-1[-]"

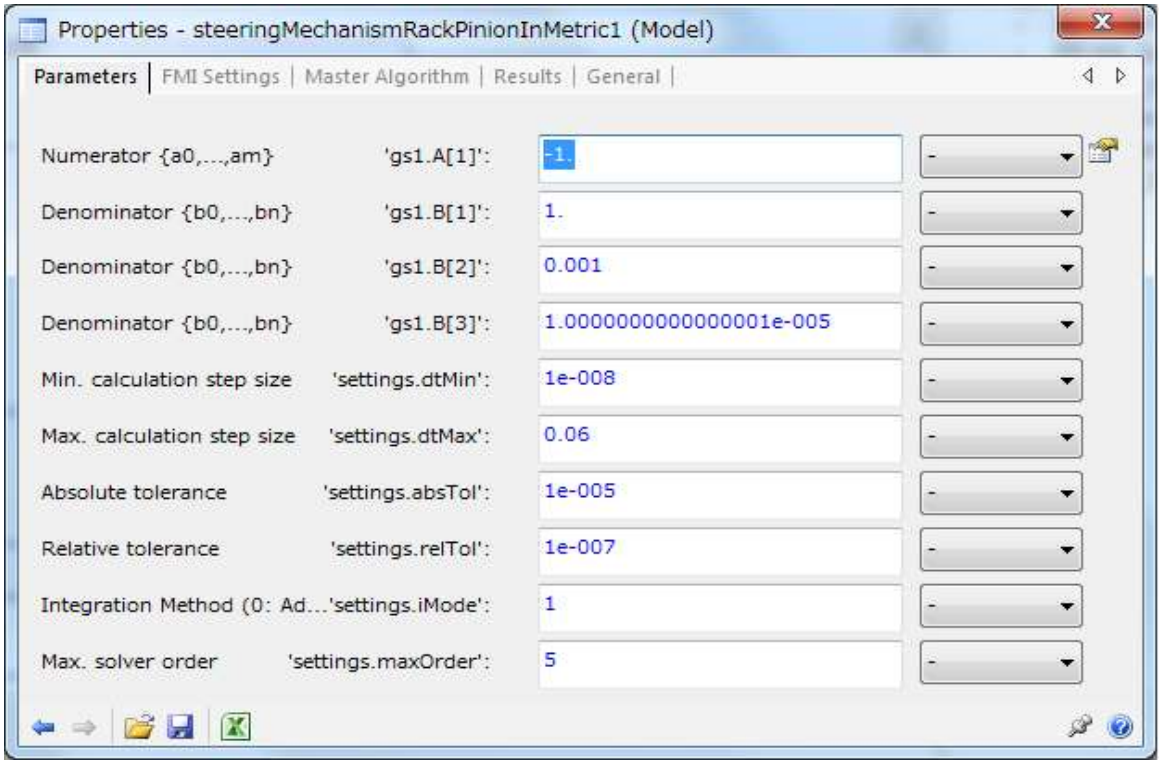


Figure 3.3.5　Steering parameter settings (example in SimulationX)

3.3.5.　Master Tool Simulation Settings

Set the simulation time to 20 seconds, the time it takes for a series of operations to be completed in the driver model.

The solver is equivalent to "ODE1" of the fixed step.

The minimum step size should be "5e-5 [s]" for the finest computation interval, with 5 FMUs.

Start Time	: 0 [s].
Stop Time	: 20 [s]
Solver, Step Sizes and Tolerances	: Fixed step Solver
Integration method	: Euler Forward (Equivalent to ODE1)
Min. Calculation Step Size	: 0.00005 (or 5e-5) [s].
Min. Output Step Size	: 0.01 [s]: Arbitrary since it is the sample time of the output result.

In the SimulationX example, this is set in SimulationControl.

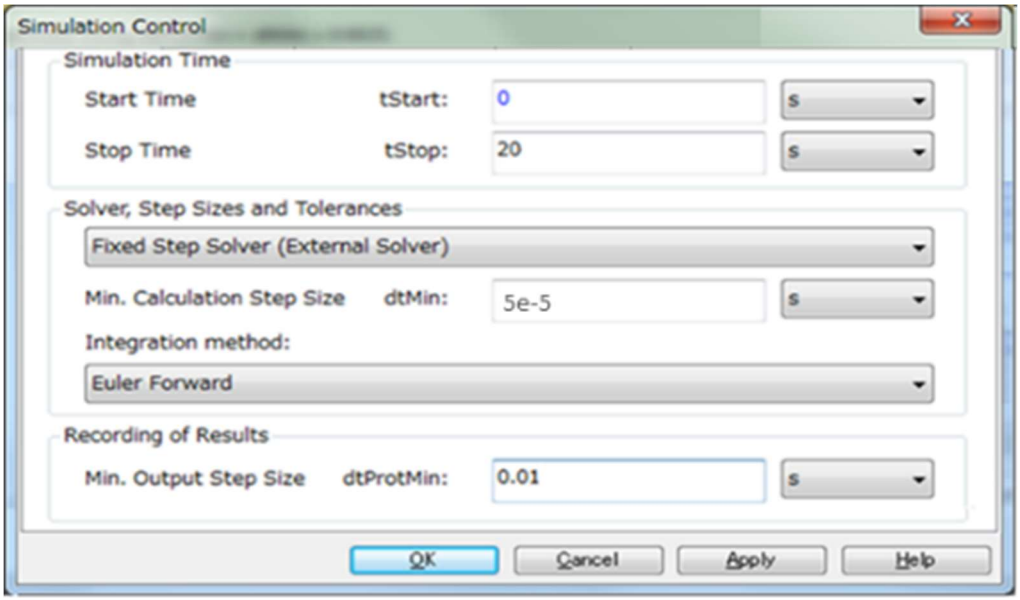


Figure3.3.6 Simulation setup (example in SimulationX)

In this example, Co-Simulation consists only of FMU on the master tool. Therefore, the difference in solver is small. However, in general use, the solver and step size should be set carefully because the model on the master tool is calculated and signals are passed between the master tool and the imported FMU.

3.3.6. Communication Step Size Communication Step Size

Co-Simulation requires a time interval setting for each FMU to receive an input signal. In the present model, Table 3.3.2 shows the settings for all FMUs.

Table 3.3.2 Communication Step Size for each FMU Setting of each FMU

Setting	Driver	Power Train	Drive Line	Chassis	Steering
Communication Step Size	5.00E-05	5.00E-05	5.00E-05	1.00E-03	1.00E-03

In the SimulationX example, this is set in the Master Algorithm tab. Set each FMU in the same way.

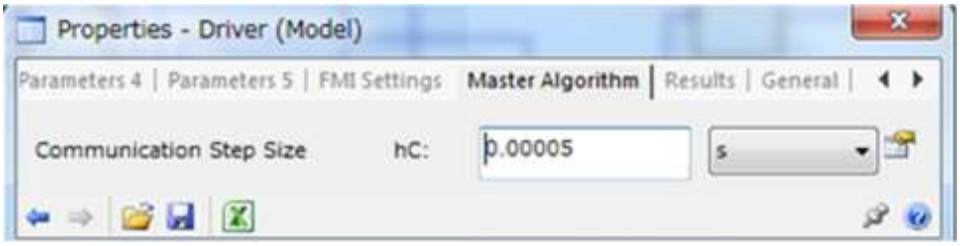


Figure 3.3 .7 Communication Step Size setting (example in SimulationX)

Communication Step Size is different for each tool, so here are some typical examples.

Amesim : Co-simulation Step Size
Dymola : fmi_Communication Step Size
MapleSim : MapleSim Sync Step
Simplorer : TS Simplorer
SimulationX : Communication Step Size hC

3.3.7. Simulation Results:

We have posted some output signals from each FMU. Did you get similar results?

For details, please refer to the simulation results saved in Sample_6p2_Result.txt.

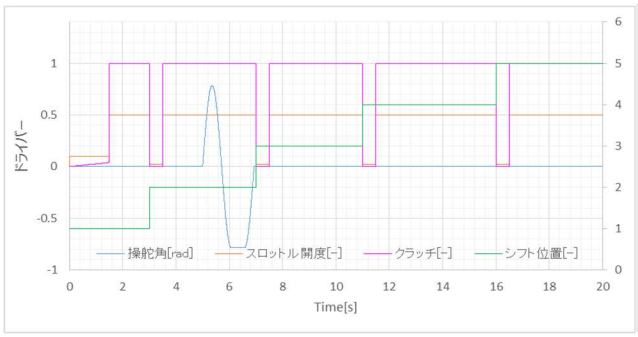


Figure3.3.8 a FMU1 (output) Driver

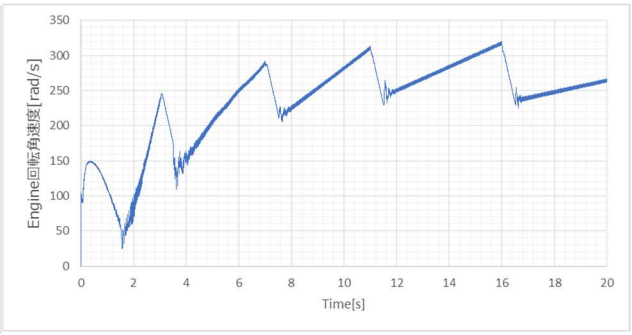


Figure3.3.8 b FMU2 (Output) Engine Speed

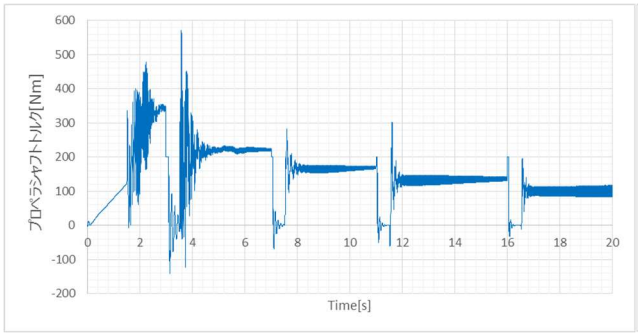


Figure3.3.8 c FMU2 (output) Prop/Shaft torque

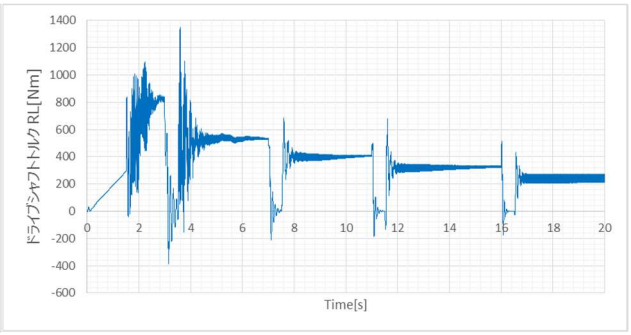


Figure3.3.8 d FMU3 (output) rear left shaft D/Shaft torque

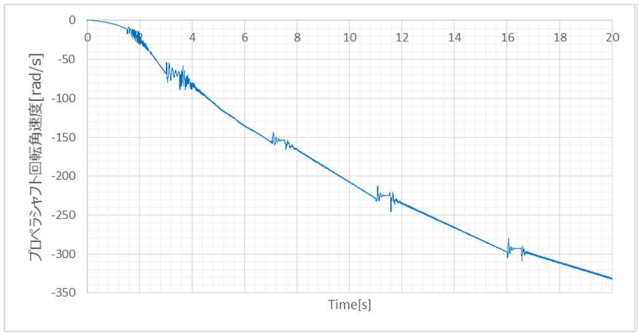


Figure3.3.8 e FMU3 (output) Prop/Shaft rotation speed

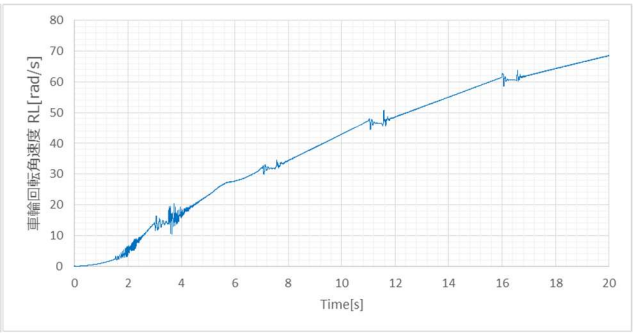


Figure3.3.8 f FMU4 (Output) Rear Left Shaft D/Shaft Rotation Speed

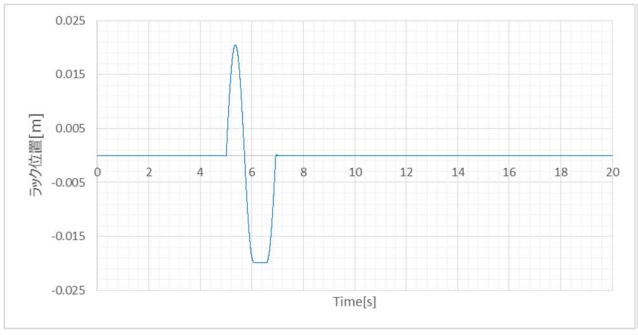


Figure3.3.8 g FMU4 (output) rack location

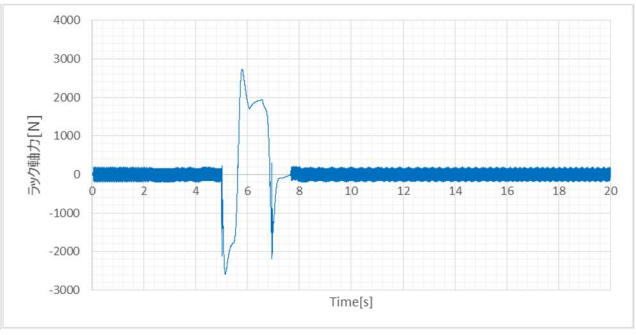


Figure3.3.8 h FMU5 (output) rack axial force

3.4. Tutorial 4: Example of a Benchmark Model Compliant with METI Guidelines Example of a Benchmark Model Compliant with

In March 2017, the Ministry of Economy, Trade and Industry's Study Group on the Use of Models in the Automotive Industry presented guidelines and released the compliant model Ver. 1.0 (compliant model is a Simulink-based model) at the same time. Please refer to the JAMBE website for details.

3.4.1. Description of Sample Models

The tutorial will now proceed with an example of our WG's benchmark model that was presented at the Society of Automotive Engineers of America Spring Conference in May 2018. [3]

METI Guidelines Seven FMUs were created based on the compliant model.

For the Driver model (Split 1) and Vehicle model, it is divided into six (Split 2 to Split 7).

Sample models can be downloaded from the Automotive Controls and Models Division Committee page of the Society of Automotive Engineers of Japan website FMI2.0 Please note that only Windows 64-bit CS is supported.

File name：Sample_6p3.zip　　：ZIP file contains the following 9 files.

FMU ：Driver.fmu, ENG_CNT.fmu, TM_CNT.fmu, PWT_PNT.fmu, CHA_PNT.fmu, ALT_CNT.fmu, ELEC_PNT.fmu

Time series data of target vehicle speed in JC08 mode　　: jc08.csv

Simulation Results　　：Sample_6p3.txt

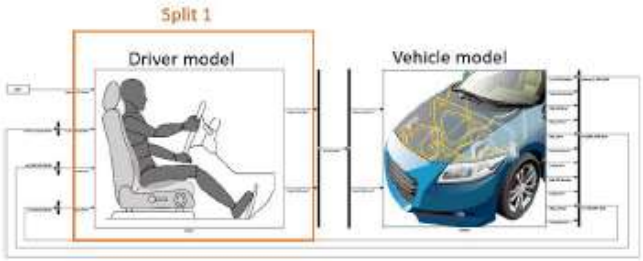


Figure 3.4.1 Model compliant with Ministry of Economy, Trade and Industry guidelines Top-level hierarchy

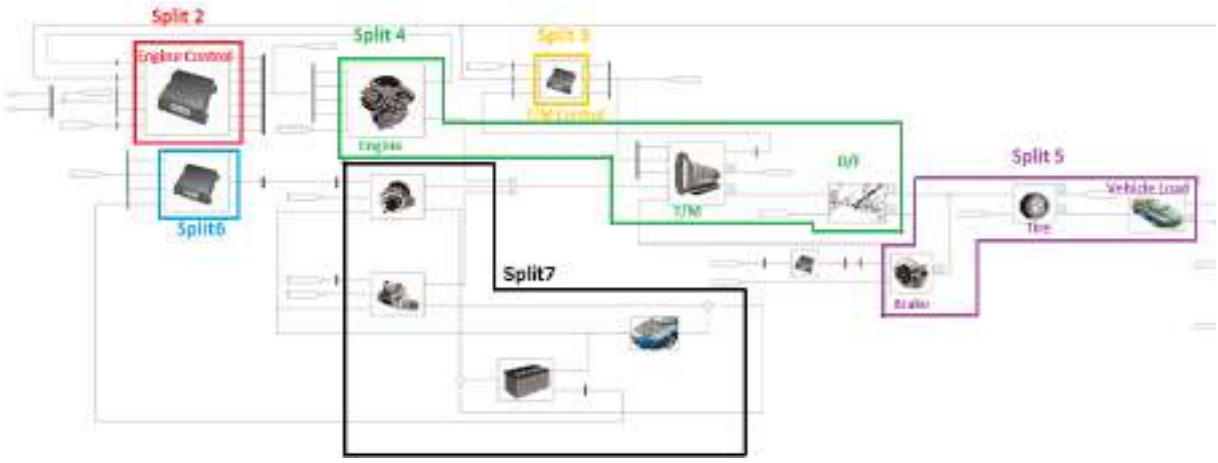


Figure3.4.2 METI Guidelines Structure of compliant vehicle model

First, import the 7 FMUs into the simulation tool to be used. The method of importing depends on the tool, such as specifying on the menu screen or dragging and dropping from Explorer, etc. Please refer to the manual.

3.4.2. Model Connections

Connect each FMU with reference to Table 3.4.1. The yellow hatched area indicates the target vehicle speed, and the green area indicates the output for monitoring.

Figure 3.4.3 is an example model connection. The left terminal of each FMU is an input and the right terminal is an output. The terminals for monitors are not used for connection. Note that the appearance of the connection varies depending on the tool used.

Table 3.4.1 Input/output signals of each FMU

FMU Number		Output Signal			Input Signal	FMU Number	FMU Name	Output Signal	
3	1	ratio_CVT	→	1	ratio_CVT	1	Driver	open_brake_per	1
2	5	n_eng_rpm_sig	→	2	n_eng_rpm_sig			open_accel_per	2
		JC08 km/h	→	3	target_v_VL_kmph				
5	2	V_PNT_kmph	→	4	v_VL_PNT_kmph				
3	2	flag_lockup	→	1	flag_Lockup	2	ENG_CNT	open_throttle_per	1
4	8	w_ENG_PNT_radps	→	2	n_eng_rpm			flag_fuelcut	2
1	1	open_brake_per	→	3	open_brake_per			flag_IdelStop	3
1	2	open_accel_per	→	4	open_accel_par			timing_ignition	4
5	2	V_PNT_kmph	→	5	v_VL_PNT_kmph			n_eng_rpm_sig	5
								flag_ON_Starter	6
4	1	n_TM_PNT_rpm	→	1	n_TM_PNT_rpm	3	TM_CNT	ratio_CVT	1
5	2	V_PNT_kmph	→	2	v_VL_PNT_kmph			flag_Lockup	2
2	1	open_throttle_per	→	3	open_throttle_per			omg_Slip_rpm	3
5	1	w_TR_PNT_radps	→	1	w_TR_PNT_radps	4	PWT_PNT	n_TM_PNT_rpm	1
3	1	ratio_CVT	→	2	ratioCVT			n_eng_rpm	2
3	2	flag_Lockup	→	3	flag_Lockup			trq_DF_PNT_Nm	3
3	3	omg_slip_rpm	→	4	omg_Slip_rpm			tScope_Fuel	4
2	1	open_throttle_per	→	5	open_throttle_per			tScope_Fuelratio	5
2	2	flag_fuelcut	→	6	flag_fuelcut			tScope_CVTLoss	6
2	3	flag_IdleStop	→	7	flag_IdleStop			tScope_trq_Flywheel2	7
2	4	timing_ignition	→	8	timing_ignition			w_ENG_PNT_radps	8
7	2	trq_ALT_PNT_Nm	→	9	trq_ALT_PNT_Nm				
7	3	trq_ST_PNT_Nm	→	10	trq_ST_PNT_Nm				
1	1	open_brake_per	→	1	open_brake_per	5	CHA_PNT	w_TR_PNT_radps	1
4	3	trq_DF_PNT_Nm	→	2	trq_DF_PNT_Nm			v_VL_PNT_kmph	2
								tScope_V_VL_PNT_mps	3
7	1	SOC_BT_PNT_Lo_PCT	→		SOC_BT_PNT_Lo_PCT	6	ALT_CNT	target_volt_ALT_V	1
2	2	flag_fuelcut	→	1	flag_fuelcut				
2	3	flag_IdelStop	→	2	flag_IdleStop				
2	5	n_eng_rpm_sig	→	3	n_eng_rpm_sig				
6	1	target_volt_ALT_V	→	1	target_volt_ALT_V	7	ELEC_PNT	SOC_BT_PNT_Lo_PCT	1
2	6	flag_ON_Starter	→	2	flag_ON_Starter			trq_ALT_PNT_Nm	2
4	8	w_ENG_PNT_radps	→	3	w_ENG_PNT_radps			trq_ST_PNT_Nm	3
								tScope_pulley1Loss	4
								tScope_SOC_Batt	5
								tScope_V_ALT	6

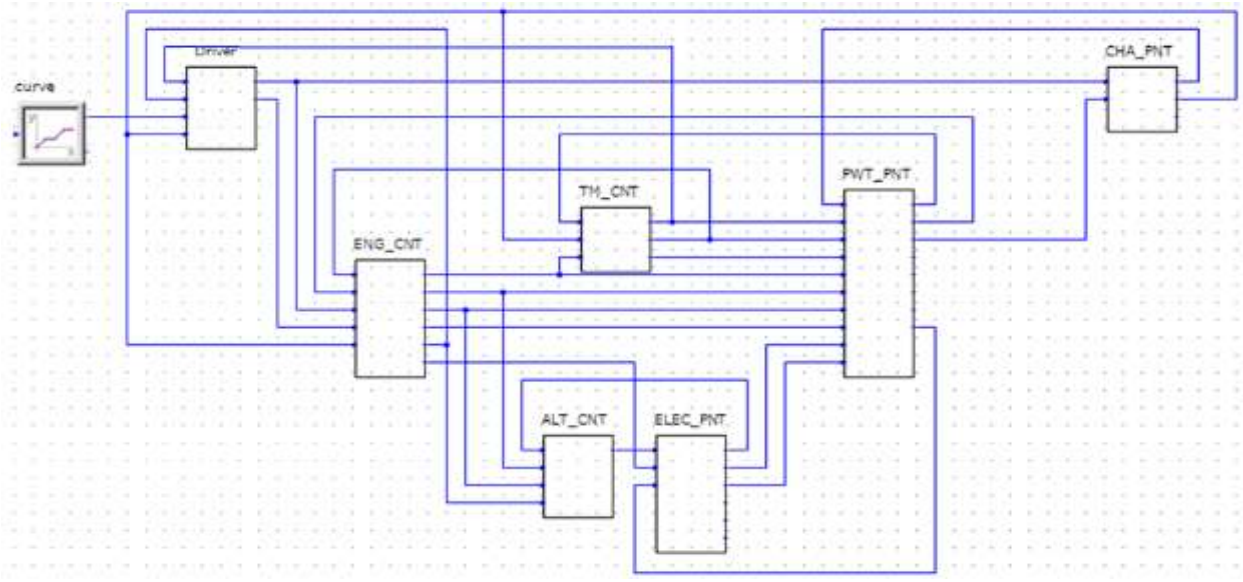


Figure 3.4.3 Example of model connections (example in SimulationX)

3.4.3. Driver Model Description

The driver model (FMU1: Driver) operates the accelerator and brake in closed loop to achieve the target vehicle speed (yellow hatched area in Table 3.4.1). This time, time series data in jc08.csv is input to drive in JC08 mode (Fig. 3.4.4). In the connection example shown in Fig. 3.4.3, the setting is made to curve.

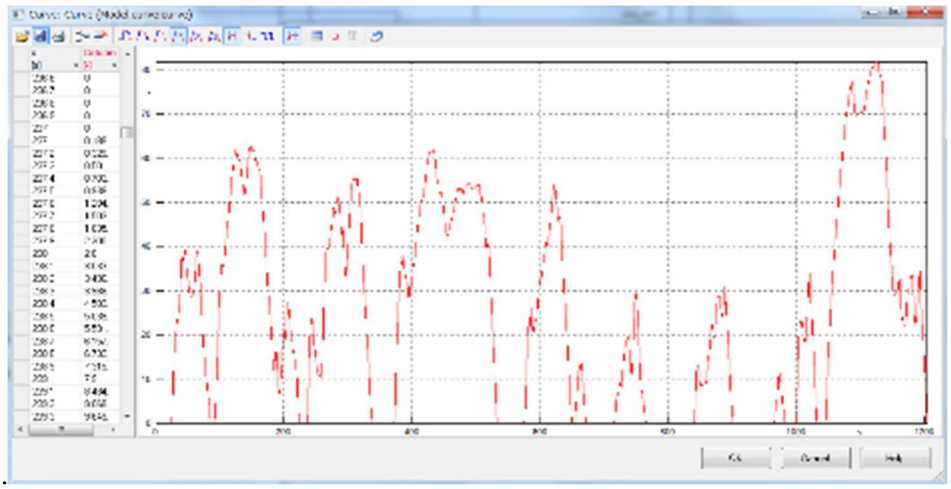


Figure3.4.4 Target vehicle speed

3.4.4. Master Tool Simulation Settings

The simulation time is set to 1204 seconds, which is the time required in JC08 mode.

The solver is equivalent to "ODE1" of the fixed step.

The minimum step size should be "2.5e-3 [s]" to match the minimum calculation interval for each FMU used.

Start Time	: 0 [s].
Stop Time	: 1204 [s].
Solver, Step Sizes and Tolerances	: Fixed step Solver
Integration method	: Euler Forward (Equivalent to ODE1)
Min. Calculation Step Size	: 0.0025 (or 2.5e-3)[s]

Min. Output Step Size : 0.01 [s]: Arbitrary since it is the sample time of the output result.

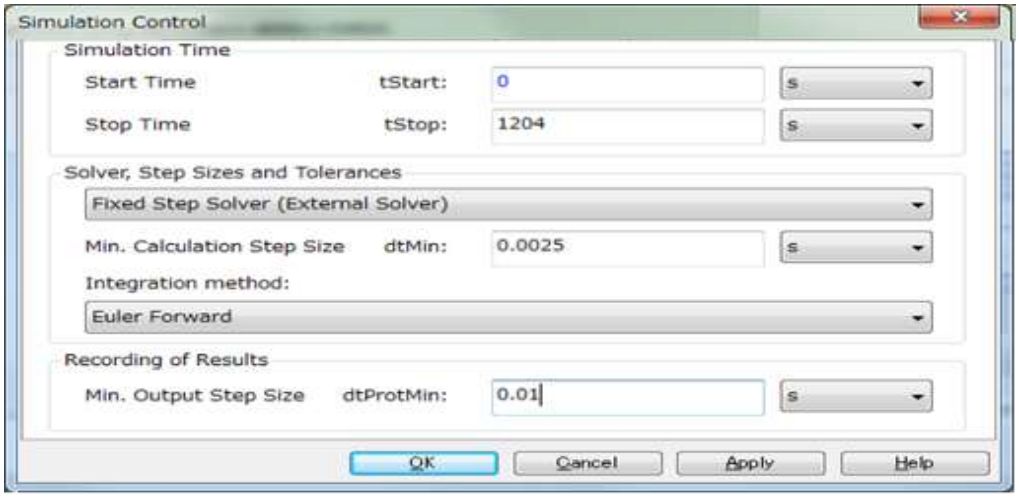


Figure 3.4.5 Simulation setup (example in SimulationX)

3.4.5. Communication Step Size Communication Step Size

Co-Simulation requires a time interval setting for each FMU to receive an input signal.

In this model, all FMUs are set at 0.0025 (or 2.5e-3) [s].

Figure 3.4.6 Communication Step Size setting (example in SimulationX)

3.4.6. Simulation Results

Some output signals from FMU are listed.

For details, please refer to the simulation results saved in Sample_6p3.txt.

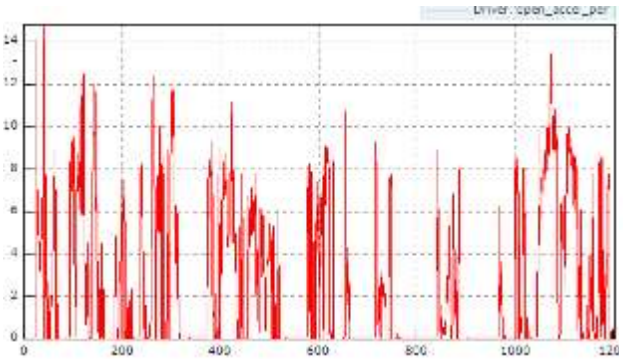


Figure 3.4.7 a FMU1 open_accel_per

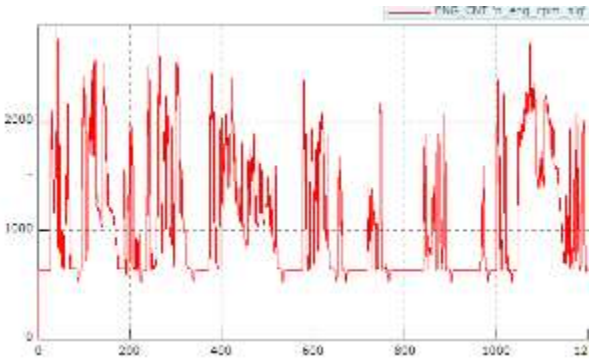


Figure 3.4.7 b FMU2 n_eng_rpm_sig

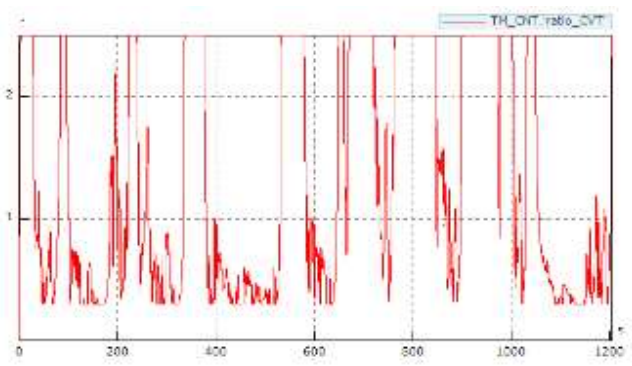


Figure 3.4.7 c FMU3 ratio_CVT

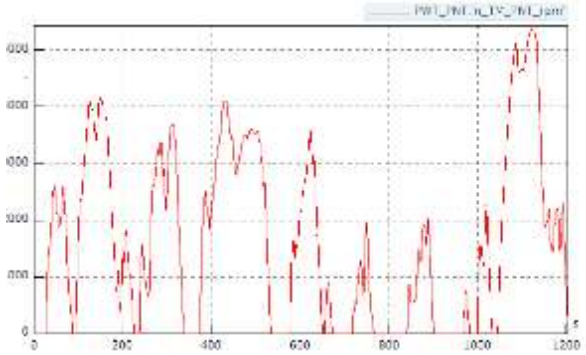


Figure 3.4.7 d FMU4 n_TM_PNT_rpm

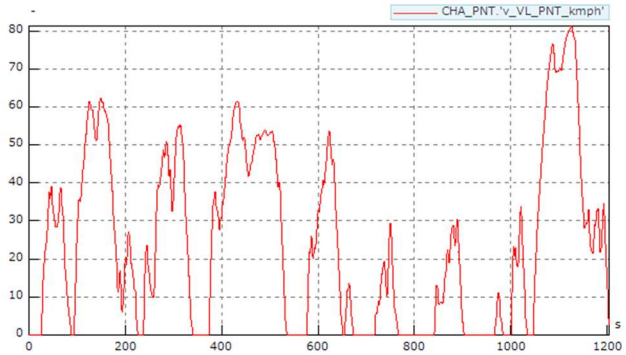


Figure 3.4.7 e FMU5 v_VL_PNT_kmph

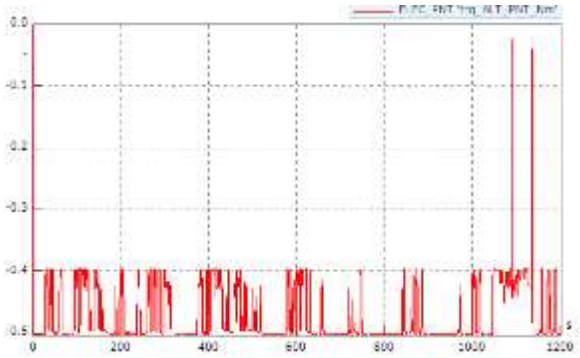


Figure 3.4.7 f FMU7 trq_ALT_PNT_Nm

References

[1] Yosuke Ogata, Shintaro Murakami, et al: A study on simulation stability in FMI, Model Exchange, and Co-simulation, Society of Automotive Engineers of Japan, 2018 Spring Meeting S4-4, (2018)

[2] Yutaka Hirano, Junichi Ichihara, etal.: Toward the actual using FMI in practical use cases in the Japanese automotive industry[p195-203],Program of the 2ndJapanese ModelicaConference, (2018)

[3] Jun-ichi Ichihara, Haruki Saito, et al: Introduction of Model Connection Activities on Co-Simulation with FMI by Multiple Modeling Tools (2nd Report), Society of Automotive Engineers of Japan, 2018 Spring Meeting S4-2, (2018)