FMI 活用ガイド（付録）

Ver.2.0.1

2024 年 6 月 1 日

Revision	日付	主な内容
1.0.0	2018/10/01	Ver1.0.0 リリース
1.0.1	2018/11/01	6 章：6.2.4 を追加、6.2.5 の最少計算時間の誤記を修正
2.0.0	2023/10/31	Ver2.0.0 リリース
2.0.1	2024/6/1	Ver2.0.1 リリース

目次

第 1 章　FMI 対応ツールの紹介 8
1.1. ダッソーシステムズ　Dymola 8
1.1.1. FMU の作成 8
1.1.2. FMU のインポートと実行 9
1.2. アンシス　Twin Builder 13
1.2.1. FMU の作成（Modelica） 13
1.2.2. FMU の作成（サブシート形式） 14
1.2.3. FMU の読み込み 16
1.2.4. その他 17
1.3. MathWorks　Simulink 18
1.3.1. FMU の作成 18
1.3.2. FMU のインポートと実行 20
1.4. Altair Twin Activate 22
1.4.1. FMU の作成 23
1.4.2. FMU のインポートと実行 25
1.5. dSPACE　SystemDesk および VEOS 28
1.5.1. SystemDesk による FMU 生成 28
1.5.2. VEOS による FMU シミュレーション 32
1.5.3. VEOS による FMI3.0 を活用したシミュレーション 33
1.6. ESI SimulationX 36
1.6.1. FMU の作成 36
1.6.2. FMU のインポート 41
1.6.3. FMU の実行 42
1.7 Gamma Technologies　GT-SUITE 44
1.7.1 GT-SUITE へのインポート 44
1.7.2 GT-SUITE からのエクスポート（FMU 生成） 47
1.8 ETAS　VECU-BUILDER 52
1.8.1 FMU の作成 52
1.8.2 FMU のインポートと実行 54

1.9. モデロン　Modelon Impact 56
1.9.1. FMU の作成 56
1.9.2. FMU のインポートと実行 57
1.9.3. OPTIMICA Compiler Toolkit の FMI サポート機能 59
1.10. VenetDCP 62
1.11. OpenModelica 65
1.11.1. FMU の作成 65
1.11.2. FMU のインポート 68
1.11.3. FMU の実行 69
1.12. FMPy 70
第 2 章　FMI 活用事例 72
2.1. ハイブリッド航空機に関する **FMI** 活用モデル 72
2.1.1. サンプルモデルの説明 72
2.1.2. モデルの概要 73
2.1.3. 航空機の飛行データ 77
2.1.4. マスタツールのシミュレーション設定 77
2.2. 1D-CAE と 3D-CAE の連成 80
2.2.1. 調査の目的（2020 年～2023 年） 80
2.2.2. 対象ツール種別の定義とツールの傾向 80
2.2.3. 3DCAE 連携の実態 81
2.2.4. 3D CAE 連携の効果と課題 82
2.3. C/C++, Python モデルと商用シミュレーションツールの連成 84
2.3.1. ユースケース 84
2.3.2. ユースケース 1 の実現方法 84
2.3.3. ユースケース 2 の実現方法 86
2.4. FMI に準拠した車載通信プロトコル(CAN)の実現 88
2.4.1. 進化する E/E アーキテクチャ開発が抱える課題 88
2.4.2. マルチデバイス用協調シミュレーション環境の概要 88
2.4.3. CAN モデル同士の接続のための施策 89
2.4.4. マルチデバイス用協調シミュレーション環境の開発状況 90

第 3 章　チュートリアル 91
3.1. チュートリアル１：FMI を使ってみよう 91
3.1.1. サンプルモデルを準備しよう 91
3.1.2. Model Exchange でシミュレーションしよう 92
3.1.3. Co-Simulation でシミュレーションしよう 94
3.2. チュートリアル２：FMI を作ってみよう 96
3.2.1. サンプルモデルを使って FMU を生成してみよう 96
3.2.2. 必要なパラメータのみ表示しよう 99
3.3. チュートリアル３：当 WG ベンチマークモデルの例 101
3.3.1. サンプルモデルの説明 101
3.3.2. モデルの接続 102
3.3.3. ドライバモデルの説明 102
3.3.4. FMU パラメータの設定 103
3.3.5. マスタツールのシミュレーション設定 104
3.3.6. Communication Step Size の設定 105
3.3.7. シミュレーション結果： 106
3.4. チュートリアル４：経産省ガイドラインに準拠したベンチマークモデルの例 107
3.4.1. サンプルモデルの説明 107
3.4.2. モデルの接続 109
3.4.3. ドライバモデルの説明 110
3.4.4. マスタツールのシミュレーション設定 110
3.4.5. Communication Step Size の設定 111
3.4.6. シミュレーション結果 111
参考文献 112

著作権について

本ガイドの著作権は、自動車技術会 自動車制御とモデル部門委員会 FMI 活用・展開 WG に帰属します。本ガイドは自動車開発の手法や品質を保証するものではありません。また、掲載事項は予告なしに変更または廃止される場合があります。実活用に対しては各ビジネスモデルに合わせた適用判断をお願いします。

本ドキュメントの取扱いについて

本ガイドは、非営利目的、または利用者内部で使用する場合に限り、複製が可能です。また、本ガイドを引用する場合は、本ガイドからの引用であることを明示し、引用された著作物の題号や著作者名を明示する等の引用の要件を満たす必要があります。

本ガイド作成にあたり、自動車技術会 自動車制御とモデル部門委員会 FMI 活用・展開 WG に参加、協力頂いた個人、企業・団体は以下の通りです。

江嶋 睦仁	株式会社 IDAJ
市原 純一	AZAPA 株式会社
守屋 一成	AZAPA 株式会社
関末 崇行	アンシス・ジャパン 株式会社
岩ヶ谷 崇	サイバネットシステム株式会社
緒方 洋介	シーメンス 株式会社
VIRY、 Guillaume	ダッソー・システムズ 株式会社
都築 勝也	dSPACE Japan 株式会社
吉松 俊	dSPACE Japan 株式会社
内田 裕美	株式会社 電通総研
上山 慶一	株式会社 デンソー
三輪 修也	株式会社 デンソー
荒木 大	東芝デジタルソリューションズ株式会社
平野 豊	HIRANO Research Lab.
猿木 恭文	HEXAGON 株式会社
平井 信之	HEXAGON 株式会社
斉藤 春樹	日産自動車 株式会社
神野 英章	株式会社 本田技術研究所
平尾 俊一	株式会社 本田技術研究所
赤阪 大介	The MathWorks GK
宅島 章夫	The MathWorks GK
遠藤 駿	マツダ 株式会社
小森 賢	マツダ 株式会社
武内 宏明	マツダ 株式会社
佐藤 裕司	三菱スペース・ソフトウエア 株式会社
GAO、 Rui	モデロン 株式会社

（企業・団体名で 50 音順）

また、本ガイド作成にあたり、下記の、個人、企業・団体の皆様にもご協力をいただきました。

石川 誠司	イータス株式会社
渋谷 賢佑	名古屋大学
重松 浩一	名古屋大学
船橋 豊	ルネサスエレクトロニクス株式会社
3V-SG（仮想的手法を用いた制御検証研究会）	

（企業・団体名で 50 音順）

第 1 章　FMI 対応ツールの紹介

この章では、代表的なツールにおける FMI 関連機能を紹介します。

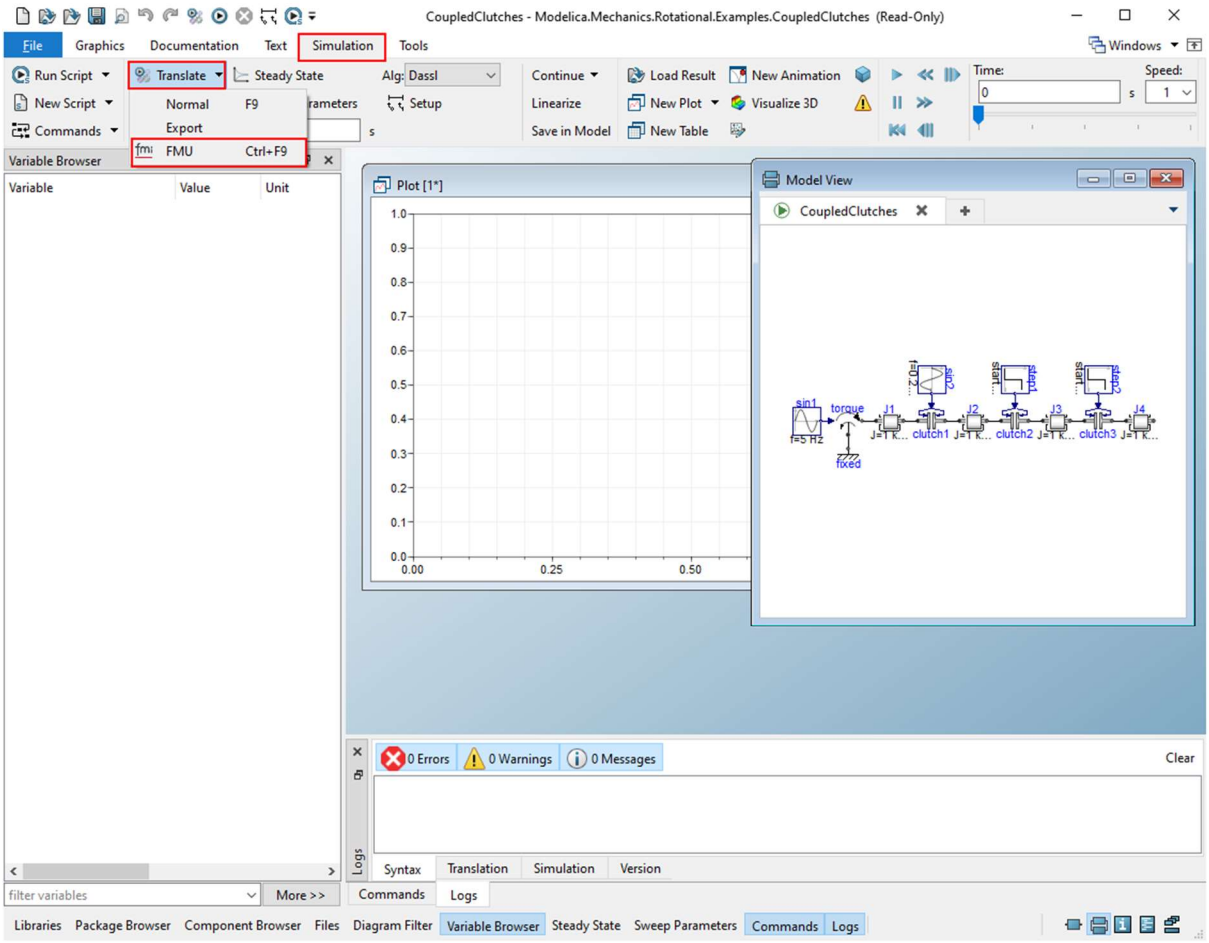
1.1. ダッソーシステムズ　Dymola

Dymola は、ダッソーシステムズ（Dassault Systemes）から Modelica 言語を使用したシミュレーションツールです。Dymola は FMI の標準に従っており、FMI に対応したモデルの作成、統合、シミュレーションを行うことができます。

- FMI 1.0、2.0、および 3.0（ベータ） をサポートします。
- Co-Simulation と Model Exchange をサポートします。
- Windows および Linux を対応しています。

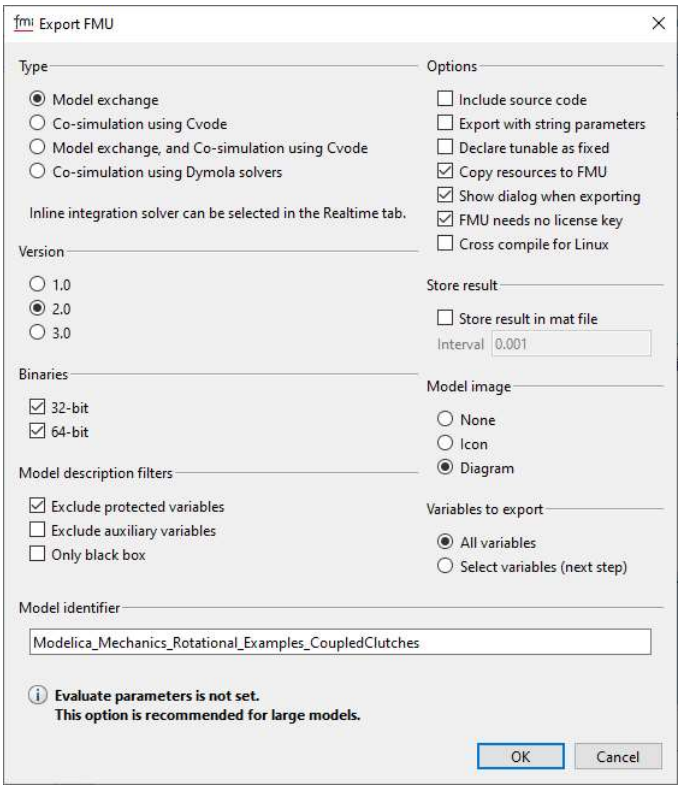
1.1.1. FMU の作成

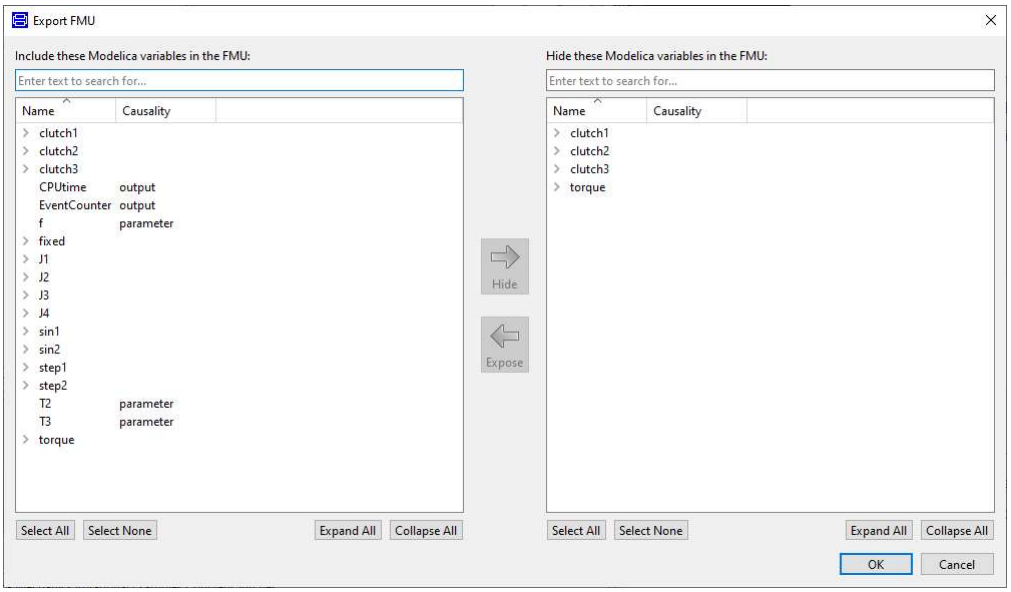
モデルから FMU を生成したければ、Simulation タブで Translate を選択します。その後、FMU を選んでください。



次の画面では、FMU 生成に関するオプションが表示されます。その内、FMU の種類（Model Exchange のみ、 Co-simulation のみ、またはハイブリッド FMU）や標準のバージョン、プラットフォームなどを選びます。Co-simulation の FMU を選んだ場合、ソルバ設定がコマンドを実行した時のソルバ状態にな

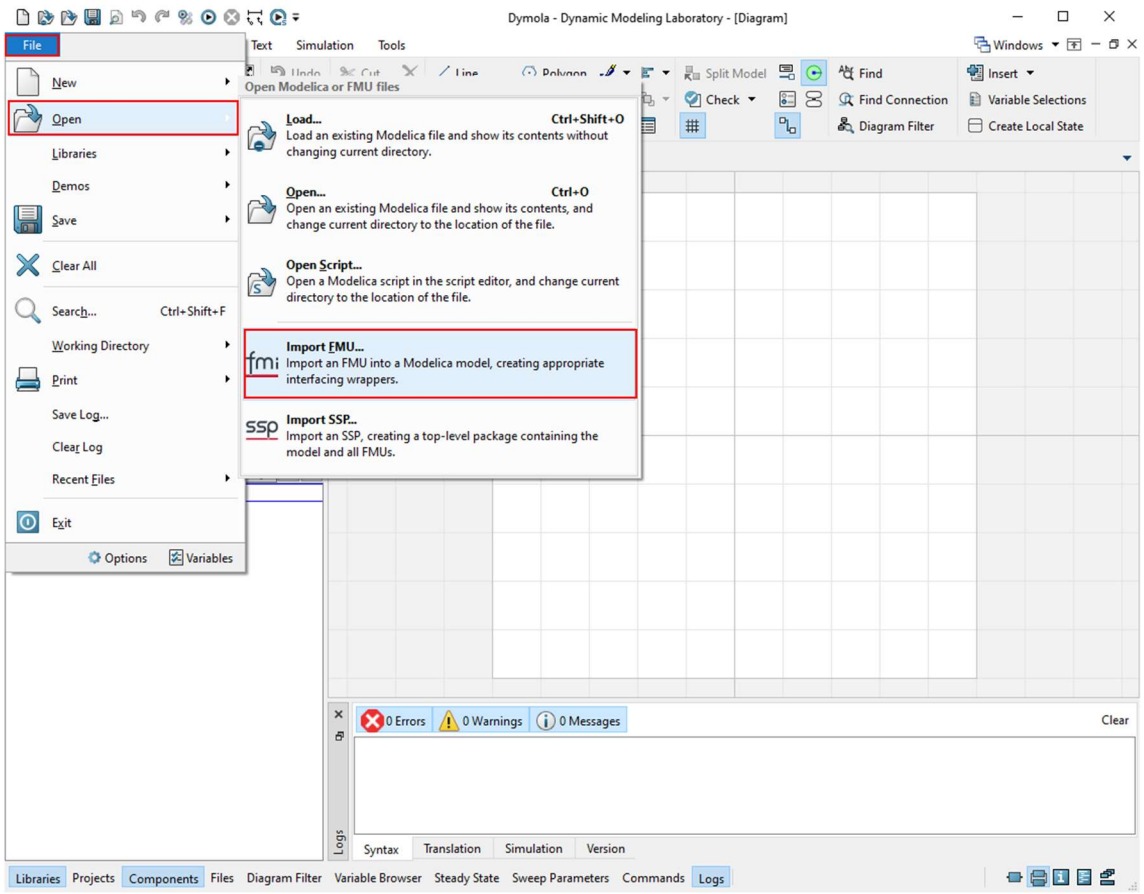
ります。必要があれば、除外する変数も決められます（例えば、I/O とパラメータしか公開したくなければ、Black-Box の FMU を選択してください）。もし、もっと細かく変数を選択したければ、Select variables をチェックしてください。そうすると、次の画面では変数選択ができます。



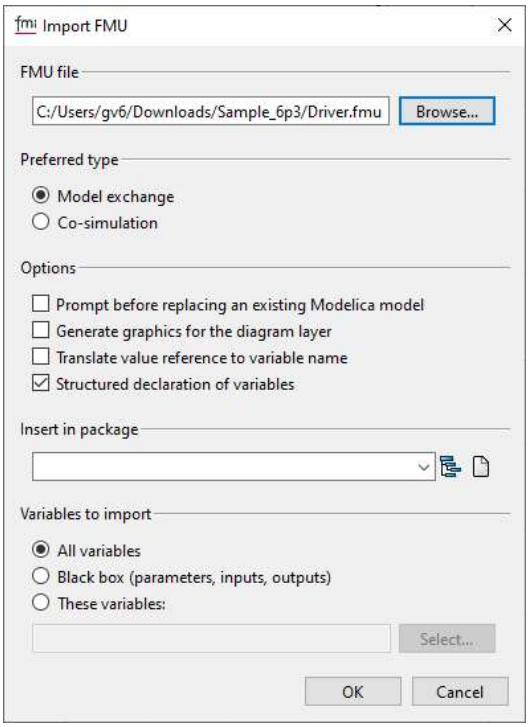


1.1.2. FMU のインポートと実行

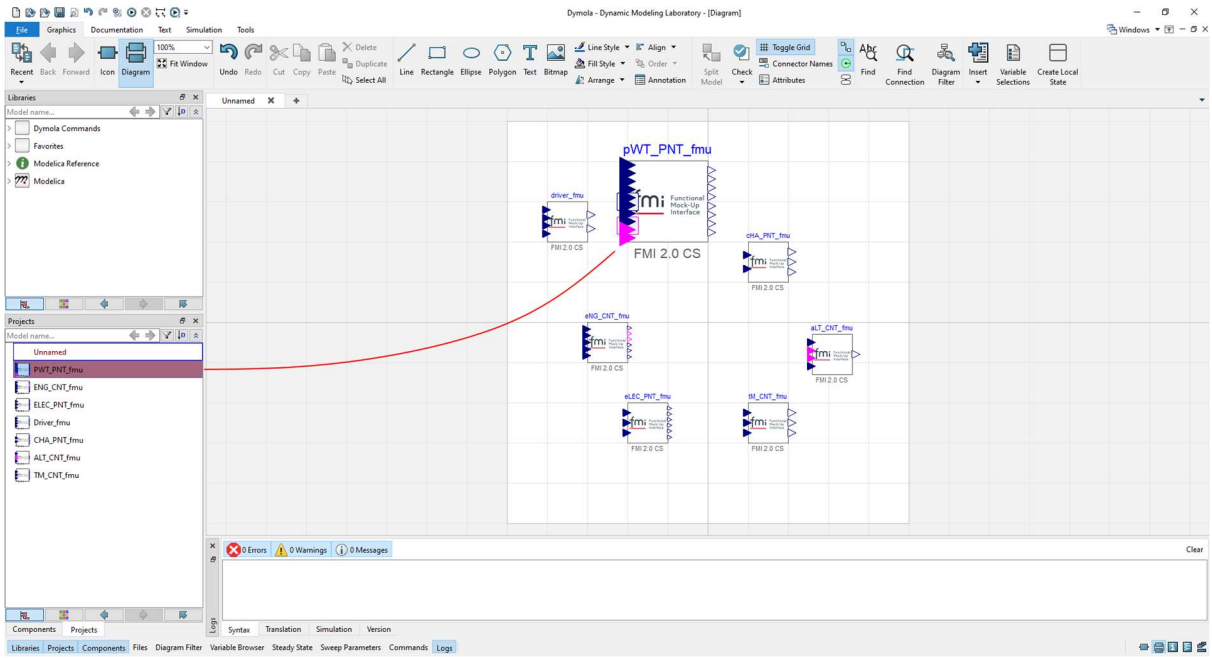
FMU をインポートしたい場合、File メニュの Open コマンドを選択して、Import FMU を選んでください。真ん中の画面に Drag & Drop でもできます。



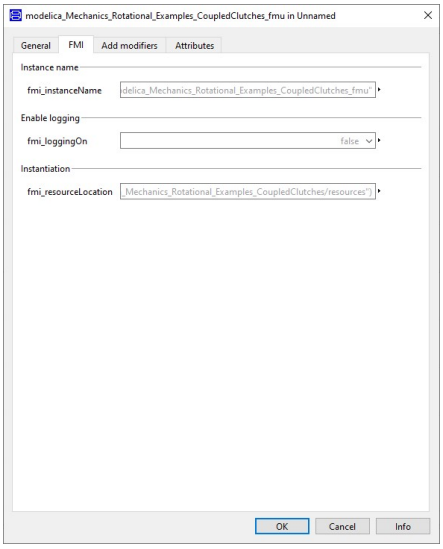
ファイルのパスを指定して、ハイブリッド FMU の場合優先的に利用されるモードを選びます。その段階で公開する変数を細かく決められます。

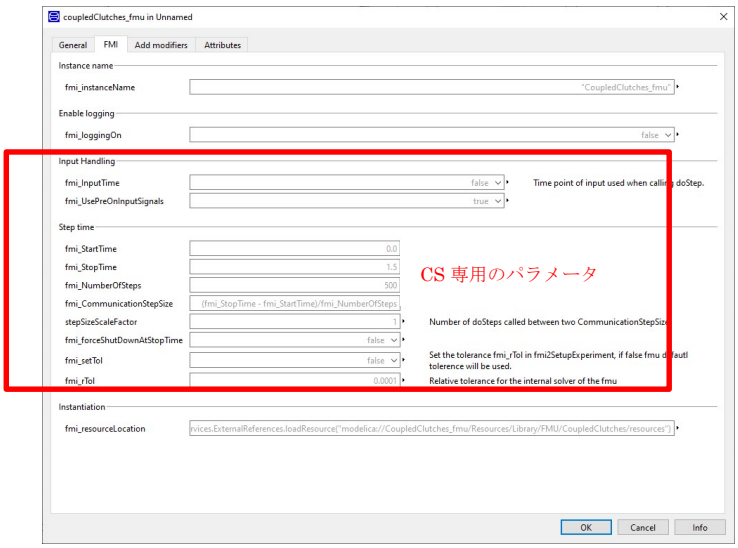


FMU を実行するには、インポートされたモデルをメイン画面に移動してください。



コンポネントにダブルクリックすると、パラメータを設定する画面が現れます。その内、FMI 専用のタブがあって、FMU のロギングやインスタンス名などを指定できます。Co-simulation の場合（右側のスクリーンショット）、開始時間、終了時間、同期化のステップもあります。



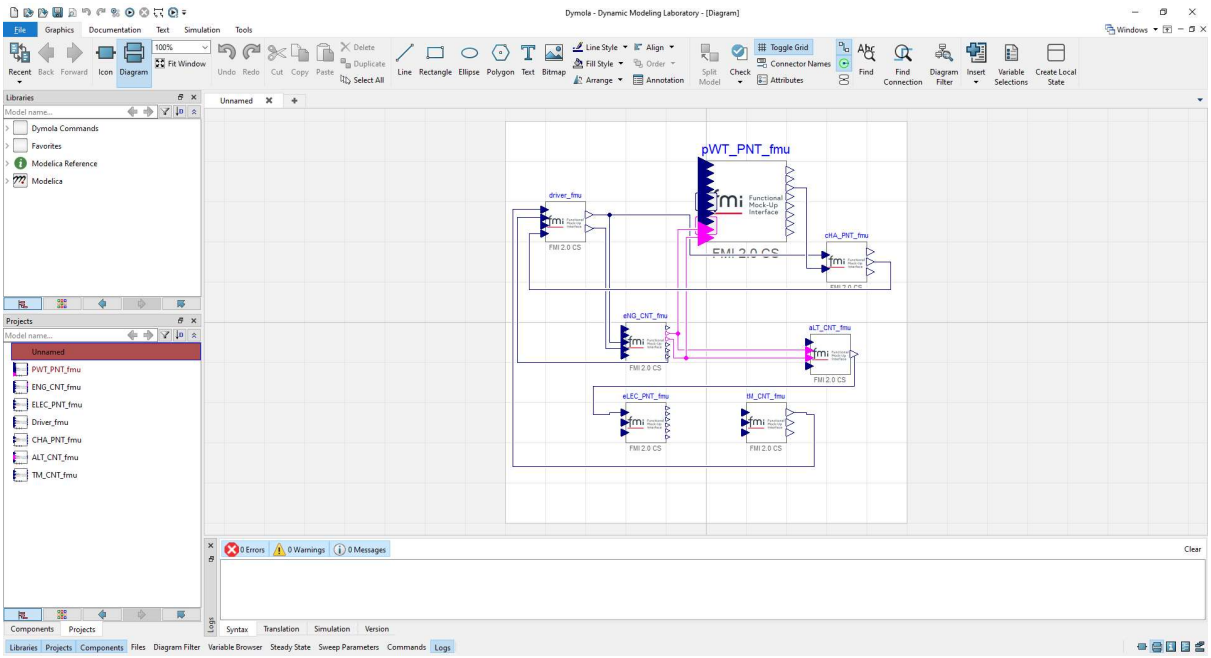


Model Exchange FMU のパラメータ　　*Co-Simulation FMU のパラメータ*

インポートされた FMU を、他の FMU および Modelica のコンポネントと接続して普通の Dymola のシミュレーションと同じように実行して結果表示と解析ができます。

Co-simulation の場合、それぞれの FMU の I/O の同期ステップが正しく設定されていることを確認してください。

複数の FMU を含めているモデルは下記のように見えます（コンポネントの接続はまだ完成していない状態）。


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
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

1.2. アンシス　Twin Builder

Twin Builder (2023.R2) が有する機能について、表 1.2.1 に示します。

FMI の作成においては Modelica ライブラリだけで構成されたモデルから FMU を作成する場合と、Twin Builder の独自ライブラリ（SML）と組み合わせてサブシート構造にて作成したモデルから FMU を作成する場合で機能や作成方法が異なります。

FMI 作成機能（Modelica）	
FMI バージョン	1.0、 2.0
形式	Model Exchange、 Co-Simulation
OS	Windows 64、 32
実行ライセンス	不要
FMI 作成機能（サブシート形式）	
FMI バージョン	2.0
形式	Co-Simulation
OS	Windows 64、 Linux 64 (Ubuntu、 RedHat)
実行ライセンス	不要（必要に切り替え可能）
FMI 読込機能	
FMI バージョン	1.0、 2.0
形式	Model Exchange、 Co-Simulation
OS	Windows 64
実行ライセンス	作成側に依存

表 1.2.1 Twin Builder FMI インタフェース機能一覧

1.2.1. FMU の作成（Modelica）

Modelica エディタを用いて作成したモデルのみを用いて FMU を作成する手順について記載します。

A）Modelica モデルの作成からコンパイル

① プロジェクトツリー Definitions/ Models を右クリックして [Add Definition] より作成するモデル名を指定し、Modelica エディタを開きます。

② Modelica ライブラリツリーが画面右側に展開されますので、ドラッグ＆ドロップで部品を配置、結線などを行い対象となるモデルを構築します。

③ リボン [Compile & Update Project]よりコンパイルを実行

④ 画面左上直下の「×」ボタンで Modelica エディタを閉じます。

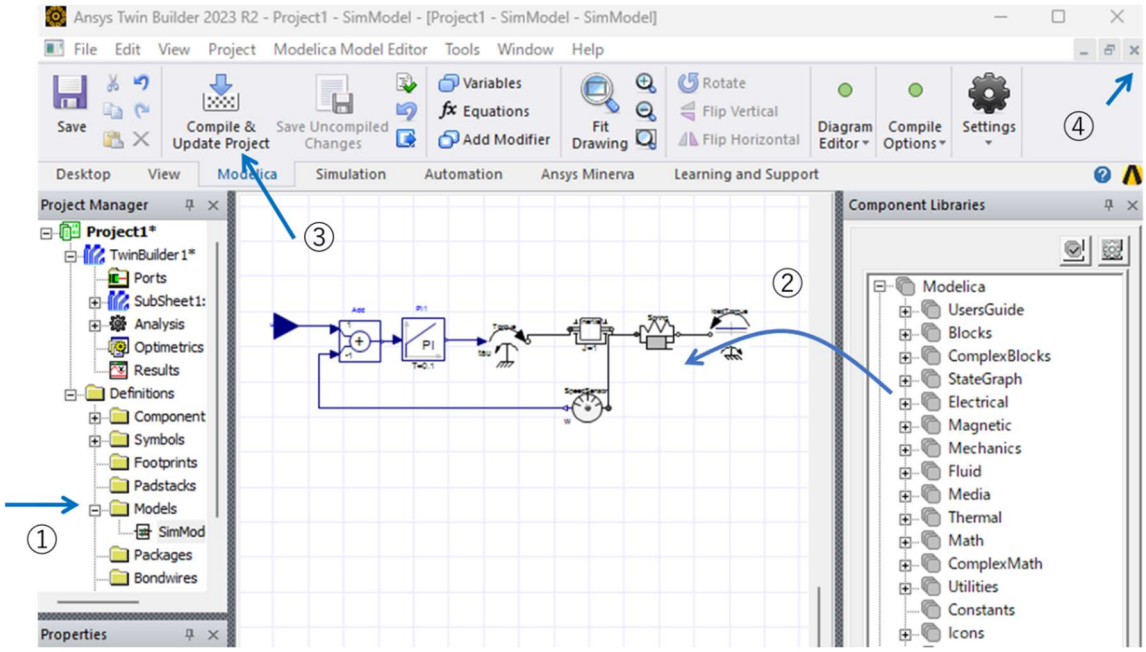


図 1.2.1　Modelica モデル編集画面

B) FMU の作成

① プロジェクトツリー Definitions/ Models より作成した Modelica モデルを右クリックして[Export as FMU] を選択します。

② FMU ファイル名、種別、対象 OS などを選択して[Export]

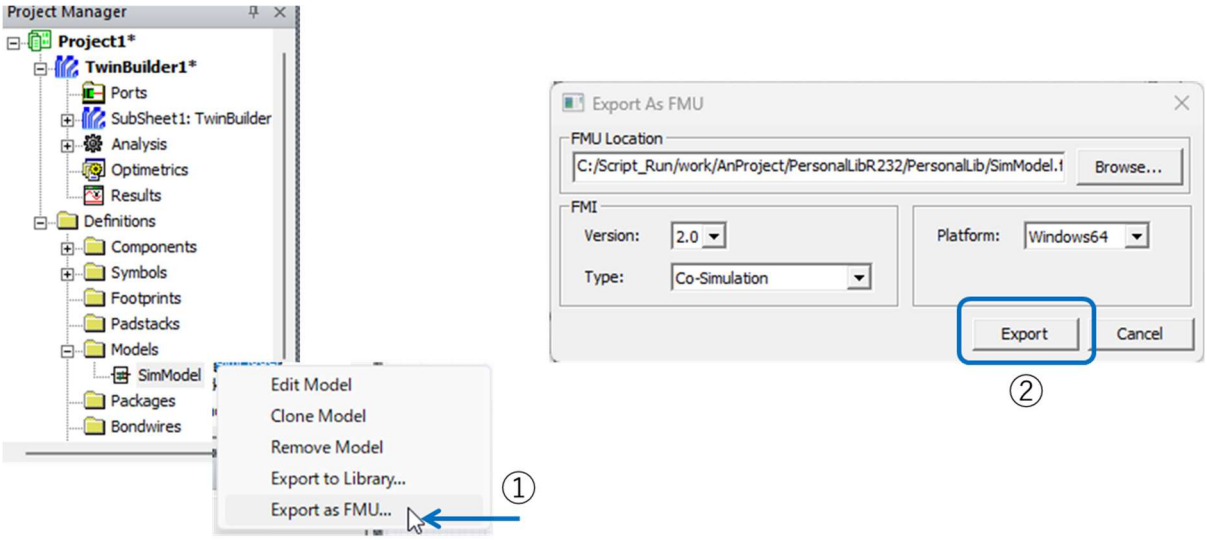


図 1.2.2　Modelica モデルからの FMU 作成画面

以上が Modelica ライブラリのみで構成されたモデルから FMU を生成する手順となります。作成した FMU モデル名は A)-①で指定する Modelica モデル名になります。

1.2.2.　FMU の作成（サブシート形式）

Simplorer 独自のライブラリモデル（SML）、Modelica モデル、C 言語モデルらをサブシート形式で構成し、その階層以下を FMU として作成する手順について示します。ここで SML モデルは主にライブラリツリー BasicElements や Multiphysics に含まれる部品を指します。

A）サブシートの作成と接続端子の準備

① リボン [Subcircuit]をクリックすると、トップレベルシート(TwinBuilder1) の下にサブシート(SubSheet1) が生成されます。

② リボンから Interface port のアイコンをクリックすると、マウスカーソルに端子のシンボルが乗りますので、適切な位置に配置します。

③ 配置した端子をダブルクリックすると、右に示すダイアログが表示され、端子名や属性を修正します。に関しては以下のようになります。

Domain：Quantity （入出力信号） Parametric（固定値パラメータ）

Type：real のみ利用可能

Direction：in （入力信号） out（出力信号）

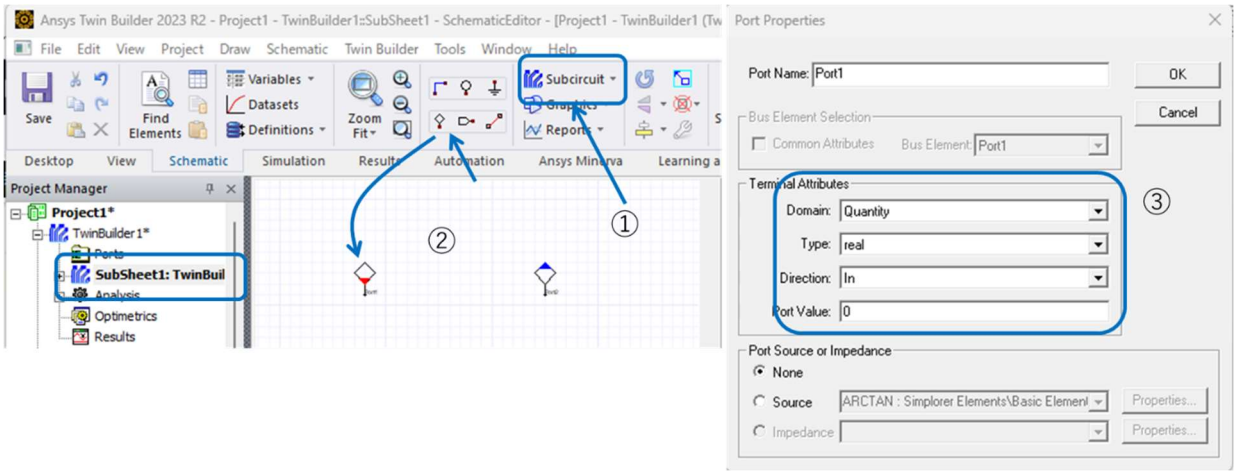


図 1.2.3　サブシートの作成

B）サブシート内部構成の作成

① 左側に表示されているライブラリツリーから部品を配置し、作成した端子と接続します。

部品には Modelica や読み込んだ FMU なども指定できます。これらはライブラリツリーの最下段 [Project Components] に存在します。

② モデルが完成したら画面の空白部を右クリックして[Pop Up]を選択し、トップレベル階層に戻ります。

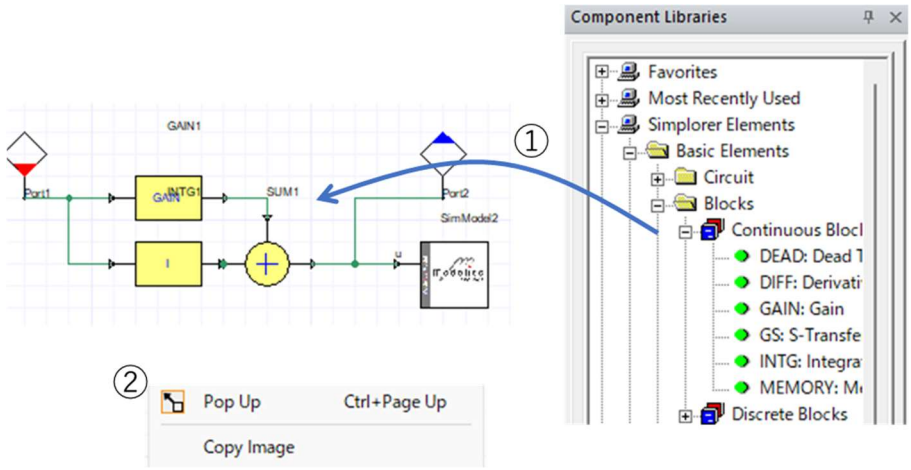


図 1.2.4　サブシート内部の作成

C）FMU の作成

① サブシートは画面左下隅に作成されます。モデルを右クリックして[Compile as co-simulation FMU]を選択すると、図右上のダイアログが開きます。

② 内部で利用している部品の dll を自動認識させるため、[Auto Detect]を選択してください。また、Linux64 にも対応する FMU を作成するには、別途 Twin Builder Linux Redistributable（無料）をインストールしておき、その保存場所を指定する必要があります。

③ コンパイル終了後、プロジェクトツリー Definitions/Models 下に FMU モデル（FMUModel_TwinBuilder2）が保存されます。これを右クリックして [Export as FMU]を選択します。

④ FMI 2.0 Co-Simulation のみが出力可能となり、[Export]で出力します。

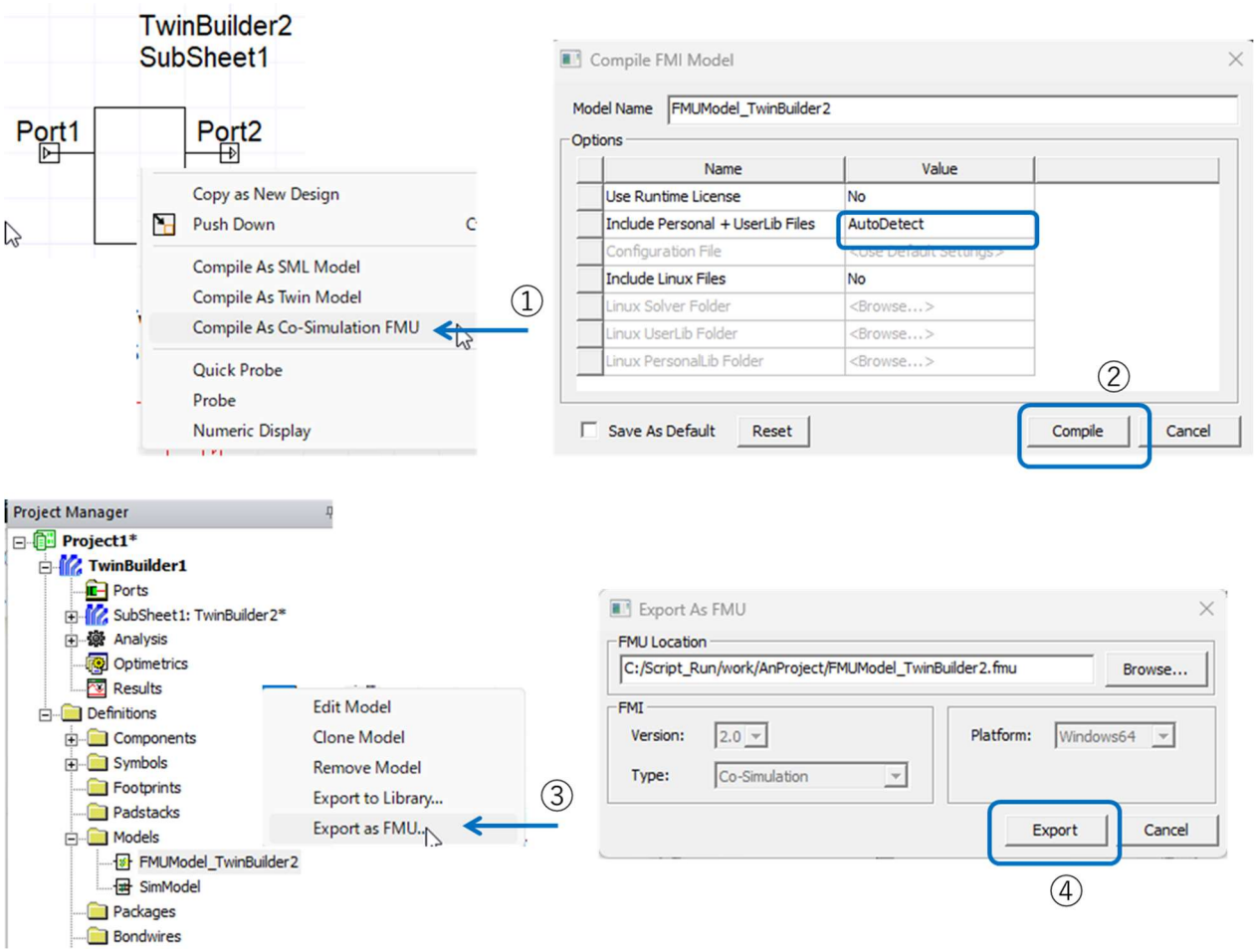


図 1.2.5　サブシート形式の FMU 出力

以上が サブシート形式で作成したモデルから FMU を生成する手順となります。C)-②にてコンパイルする際に作成した FMU モデル名が決定されます。

1.2.3. FMU の読み込み

Twin Builder は Windows64 アーキテクチャを含む FMU のみ利用できます。

A）FMU ファイルの読み込み

① トップメニュー Twin Builder/ Add Component/ Add FMU Component より、任意の FMU ファ

イルを指定します。

図左に示すダイアログが表示されますので [Import]を選択すると、カーソルに FMU モデルが乗っかりますので任意の場所にドロップします。なお、ダイアログで表示されているツリーは、結果表示や値の変更を行いたいパラメーター一覧になりますが、後で自由に追加、削除することができます。

② 配置した FMU 部品を右クリックし、[Edit Model]を選択すると、図右に示すダイアログが表示されます。

③ [Replace] ボタンで別の FMU ファイルを選択するとモデルの実体を変更できます。Model Exchange ←→Co-Simulation モデルを切り替える場合や、修正が行われた FMU モデルの更新に利用します。

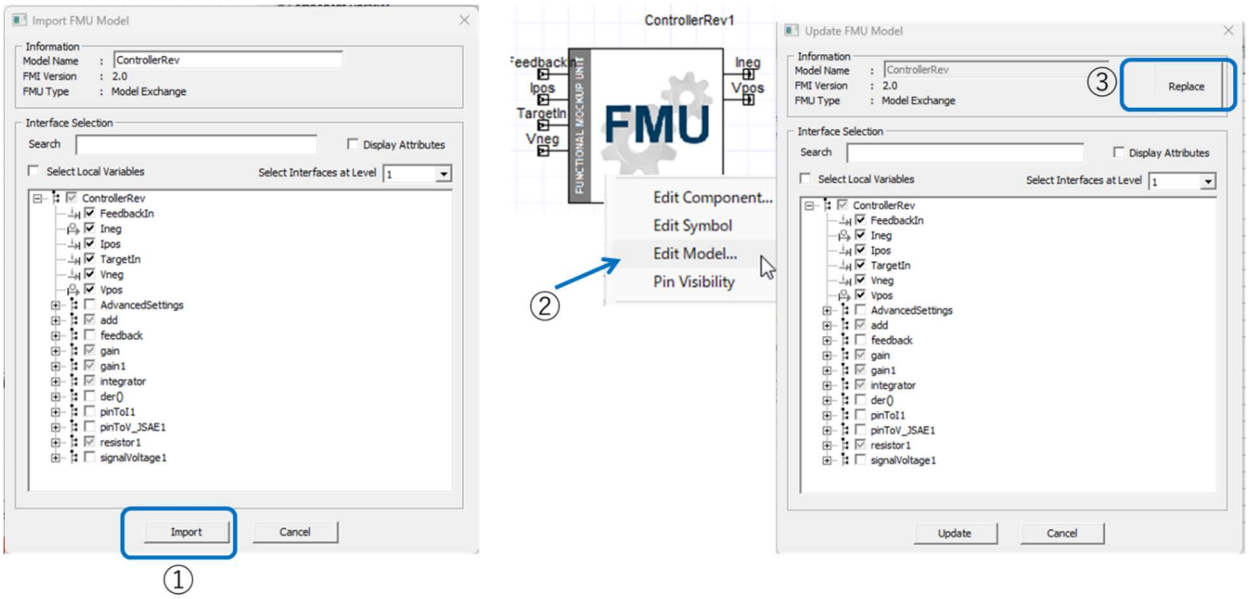


図 1.2.6　FMU の読み込み

1.2.4.　その他

Modelica に関する操作

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

サブシートに関する操作

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

FMU に関する操作詳細

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

FMU に関する操作詳細

1.3. MathWorks　Simulink

MathWorks が提供する Simulink は、FMU のインポート/エクスポートに対応しています。以下の表 1.3.1 に MATLAB バージョン R2023b におけるサポート状況についてまとめます。

表 1.3.1　Simulink の FMI サポート状況（R2023b 時点）

FMU エクスポート	
FMI バージョン	2.0, 3.0
形式	Co-Simulation (CS)
エクスポートに必要なライセンス	MATLAB, Simulink, MATLAB Compiler, Simulink Compiler
備考	• FMU エクスポートには、別途 Simulink Compiler が必要*1 • R2023b から FMI3.0 CS をサポート*3
FMU インポート	
FMI バージョン	1.0, 2.0, 3.0
形式	Co-Simulation (CS), Model Exchange (ME)
インポートに必要なライセンス	MATLAB, Simulink
備考	• Simulink の標準機能（FMU ブロック*2）で FMU インポートをサポート • R2023b から FMI3.0 CS/ME をサポート*3

*1. Simulink Compiler は R2020a にリリースされたアドオン製品です。

*2. FMU ブロックは R2017b に導入された FMU インポート用の Simulink ブロックです。

*3. R2023b 時点では、FMI3.0 の一部仕様には対応しておりません。詳細は製品ドキュメンテーションを参照ください。また、R2023b から FMU エクスポートには FMU Builder for Simulink サポートパッケージ（無償）をダウンロードし、インストールする必要があります。

1.3.1. FMU の作成

Simulink モデルから FMU をエクスポートするためには、Simulink Compiler が必要となります。以下、Simulink Compiler を用いた FMU エクスポートの方法について紹介します。なお、R2023b 時点における FMU エクスポートの制限事項として、Rapid Accelerator Mode（ラピッドアクセラレータモード）のシミュレーションモードで動作する Simulink モデルのみが対応しています。Simulink のソルバは、固定ステップソルバがサポートされています。R2023b から可変ステップソルバも FMI2.0 CS の生成に対応しています（ただし、FMU 実行先の環境に MATLAB Runtime が必要です）。詳細な制限事項については製品ドキュメンテーションを参照ください。

まず、FMU を生成する Simulink モデルを用意します。以下の図 1.3.1 に示すように、エディタ上部の [シミュレーション] タブより、[保存] > [スタンドアロン FMU] を選択します。

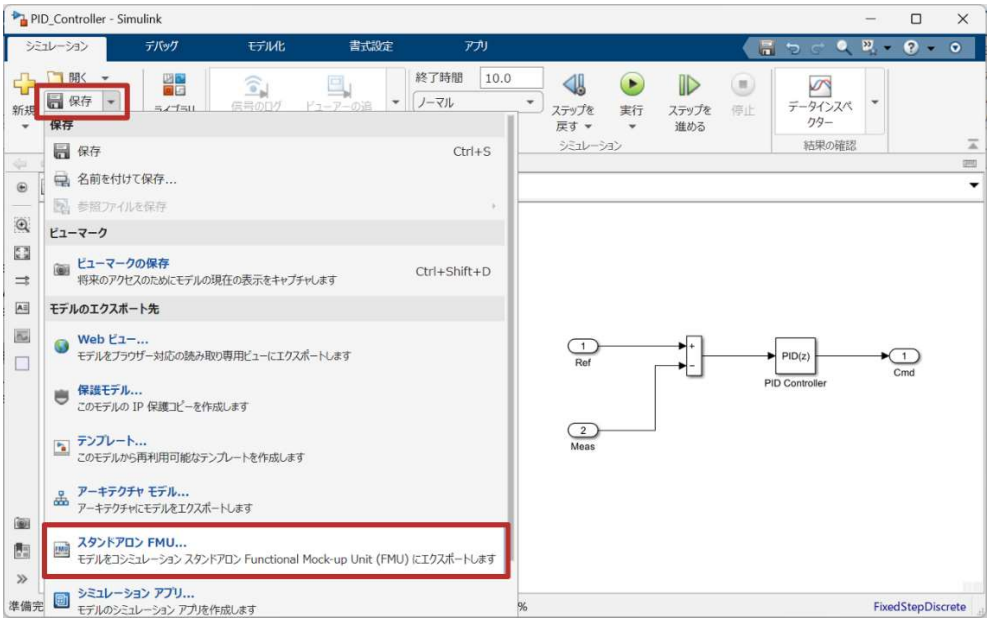


図 1.3.1　Simulink モデルから FMU エクスポート

すると、図 1.3.2 に示すようなエクスポート設定画面が表示されます。FMI バージョンやパラメータ、入出力などの設定を行い、最後に画面下の［作成］ボタンを押すと FMU が生成されます。

図 1.3.2　FMU エクスポート設定画面

上記のような GUI 操作による FMU 生成に加えて、exportToFMU2CS 関数（R2020a 導入）や exportToFMU 関数（R2023b 導入）を用いた MATLAB コマンドラインによる FMU 生成も可能です。

1.3.2. FMU のインポートと実行

Simulink には FMU をインポートするための「FMU ブロック」が用意されています。FMU ブロックは Simulink ライブラリの［Simulink Extras/FMU Import］に格納されています。以下、FMU インポートとシミュレーション実行の方法について紹介します。なお、FMU インポートは Simulink の標準機能であり、Simulink Compiler は不要です。

図 1.3.3 に示すように、Simulink エディタのキャンパス上に FMU ブロックを配置します。FMU ファイルがまだ選択されていない状態では、赤色で Unspecified FMU Import と表示されます。ブロックをダブルクリックすると、FMU ファイルを選択する画面が表示され、インポートしたい FMU ファイルをフォルダから選択します（この例では ELEC_PNT.fmu）。

図 1.3.3　Simulink モデルへの FMU のインポート

FMU ファイルを選択すると、図 1.3.4 に示すようにブロックの表示が変わり、入出力ポートが出現します。ブロックをダブルクリックすると、パラメータ設定画面が表示され、必要な設定を行います。

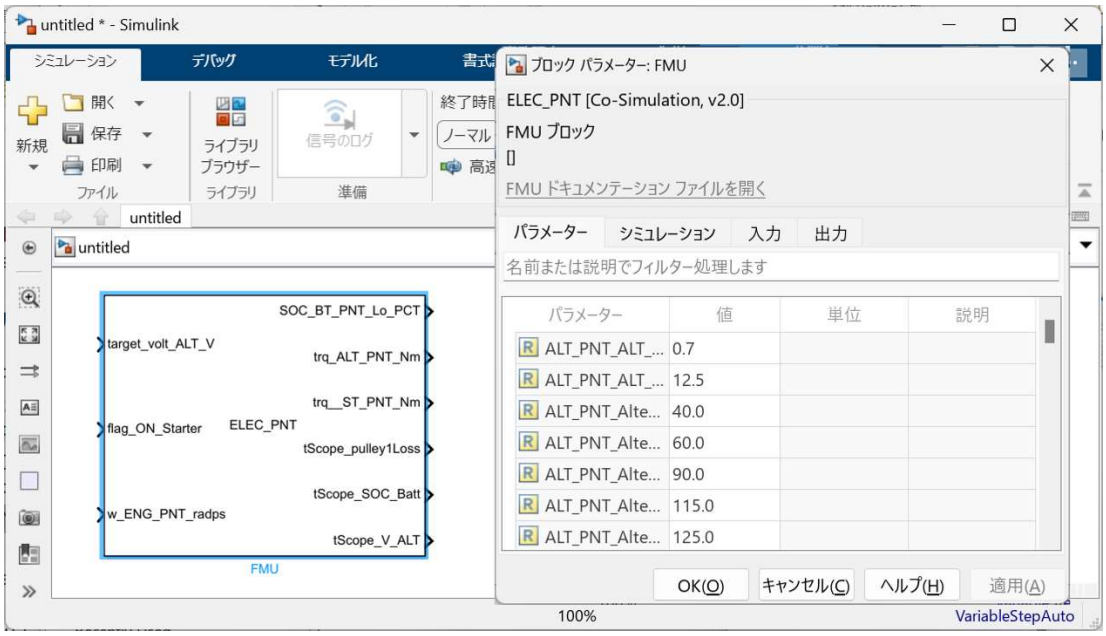


図 1.3.4　FMU ブロック

FMU ブロックは、他の Simulink のブロックと信号線で結線することができます。モデルが完成したら、図 1.3.5 に示すようにエディタ上部の［シミュレーション］タブにあるシミュレーション［実行］ボタン（緑色の丸矢印ボタン）を押すと、シミュレーションが開始されます。

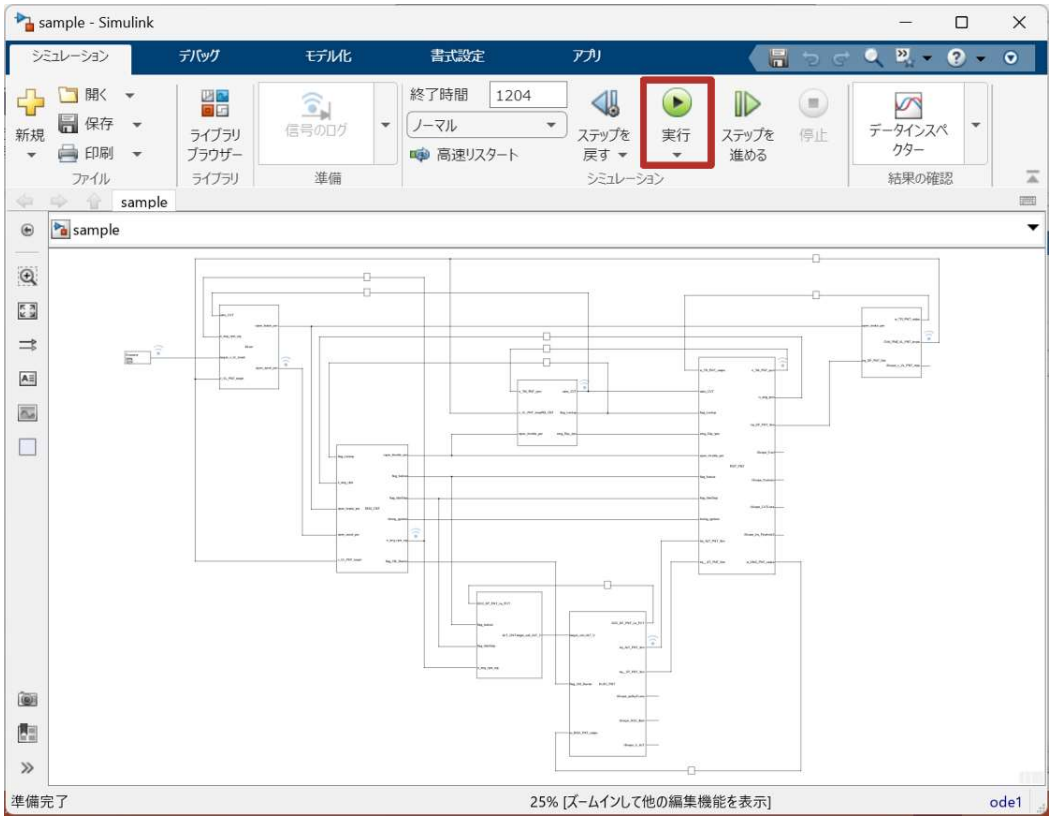


図 1.3.5　FMU を統合した Simulink モデルのシミュレーション

1.4. Altair Twin Activate

Twin Activate は包括的なモデルベース開発のためのマルチフィジックス解析が可能な統合プラットフォームです。制御の為のブロックダイアグラムライブラリを持ち、物理コンポーネントは Modelica ライブラリや他の電気・電子、油圧系システムのライブラリを活用してハイブリッドモデリングが可能です。また MotionSolve や Flux などの他の Altair 製品とのモデル連携も可能です。

Twin Activate は FMI のインポート・エクスポートに対応しており、下記の表 1.4.1 に Ver.2023 におけるサポート状況について示します。FMI 2.0 でインポート・エクスポートに対応していた機能は全て FMI 3.0 で対応が可能です。さらに、FMI 2.0 および FMI 3.0 エクスポートを使用して、インライン FMU として romAI をエクスポートすることができます。

表 1.4.1　TwinActivate の FMI インタフェース機能一覧

FMU エクスポート	
FMI バージョン	2.0, 3.0
形式	Co-Simulation (CS), Model Exchange (ME)
OS	Windows 64, Linux 64
実行ライセンス	不要 *1
備考	• Ver.2022.3 から FMI3.0 CS/ME をサポート
FMU インポート	
FMI バージョン	1.0, 2.0, 3.0
形式	Co-Simulation (CS), Model Exchange (ME)
OS	Windows 64, Linux 64
実行ライセンス	作成側に依存
備考	• Twin Activate の標準機能（FMU ブロック）で FMU インポートをサポート • Ver.2022.3 から FMI3.0 CS/ME をサポート

*1. FMU に romAI のブロックを含む場合は romAI のライセンスが必要です。

1.4.1. FMU の作成

① Twin Activate 上でモデルを作成します。FMU 作成時に入出力ポートを作製したい場合、パレッドブラウザ内 Activate ライブリの「Ports」から「Input」「Output」ブロックを配置する必要があります。

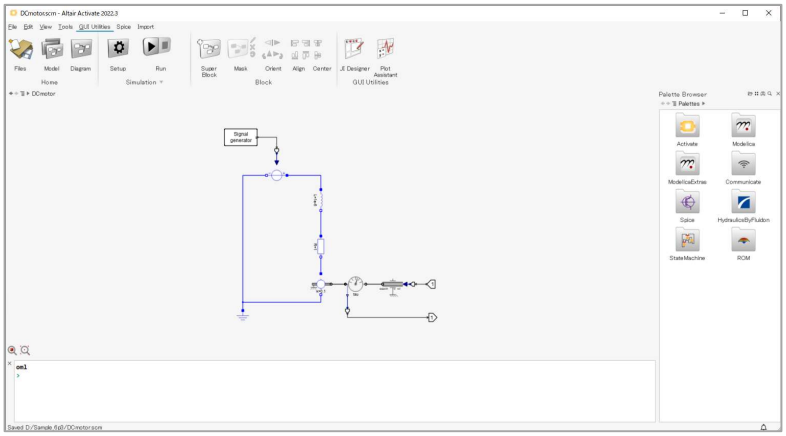


図 1.4.1　Twin Activate モデル

② 次にスーパーブロックという機能を用い、FMU にしたいモデルブロック群をグループ化します。ポートを除いたモデルを選択し、右クリックから「Super Block」を選択し作製する事が可能です。

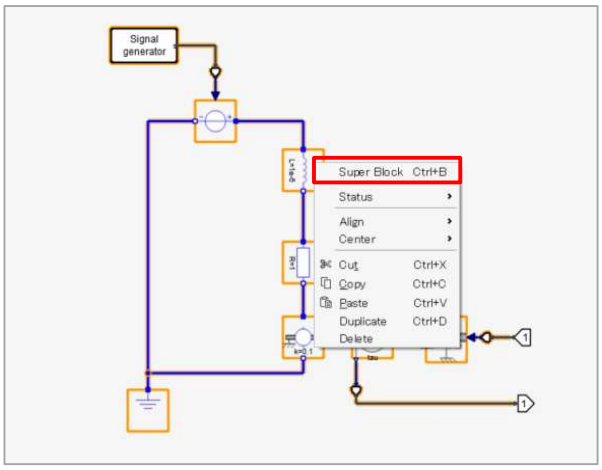


図 1.4.2　Super Block 作成

③ スーパーブロックをダブルクリックして展開し、その中の Input ポートブロックをダブルクリックしてプロパティダイアログを開きます。ダイアログ内の「Time dependency」にチェックを入れます。この項目を選択しない場合は、図 1.4.5 のダイアログの「Force inputs to always active」にチェックが入っていればエラーは発生しません。

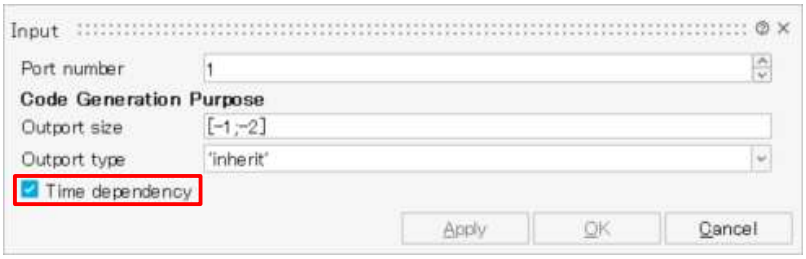


図 1.4.3　Input ポートブロック設定

FMU 作成時にモデルのパラメータを隠蔽したい場合は、この段階でスーパーブロックのパラメータに隠蔽処理を行って下さい。

④　作製したスーパーブロックを選択し、「Tools」の「Code Generation and Export」を選択します。

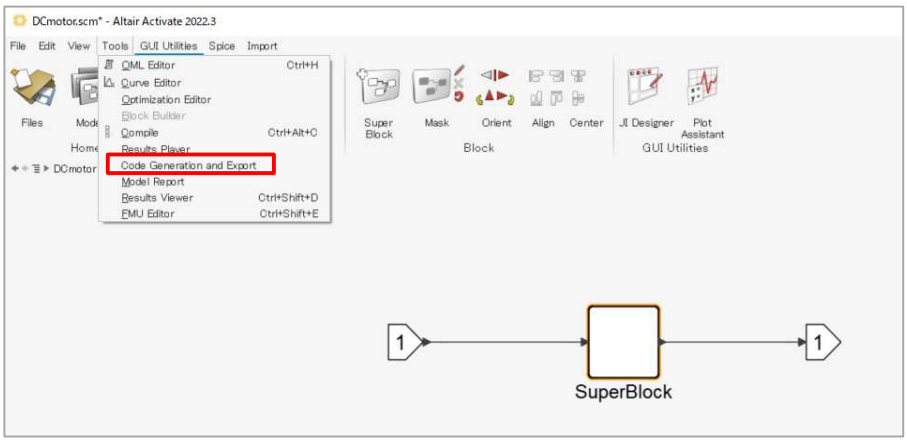


図 1.4.4　Code Generation and Export の選択

このダイアログで「Target」を「FMU」とすると「FMU type」「FMU version」など生成に関するオプションが表示されますので、必要に応じて選択してください。

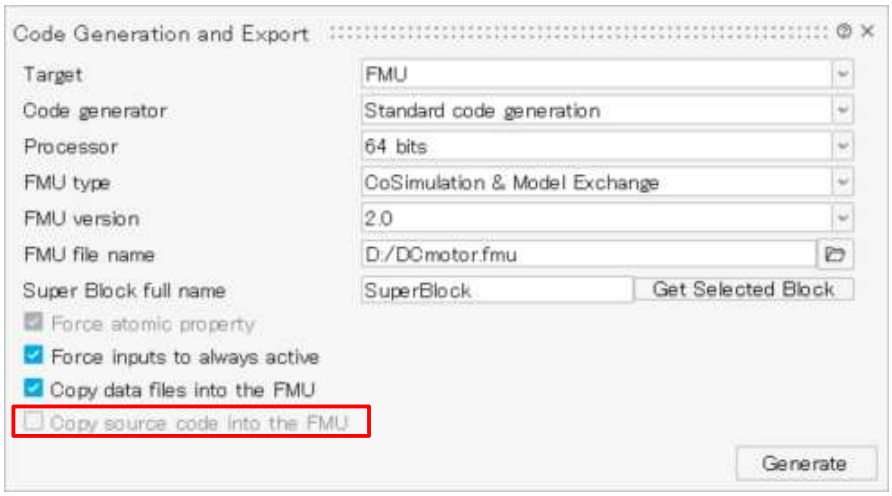


図 1.4.5　FMU 生成オプション

「FMU type」は「Model Exchange」「CoSimulation」「CoSimulation & Model Exchange」が、「FMU version」は 2.0 と 3.0 が選択可能です。
「Code generator」は Activate ブロックのみ、もしくは他の Modelica ブロックなどとのハイブリットモデルで Windows/Linux で運用する場合は「Standard code generation」を選択します。Activate モデルだけで構成されハードウェア実装での運用を行う場合は「Inlined code generation」を選択します。ただし「Inlined code generation」では非対応ブロックが存在します。この時、左下の「Copy source code into the FMU」にチェックを入れる事で、FMU 内にソースコードをコピーできます。

⑤　適切な項目を選択後、「Generate」ボタンを押下すると、「FMU file name」に指定したディレクトリに FMU が生成されます。

1.4.2. FMU のインポートと実行

① パレッドブラウザ内 Activate ライブリの「CoSimulation」から「FMU」ブロックを選択し、モデリングウィンドウに配置します。

図 1.4.6　FMU ブロックの配置

② FMU ブロックをダブルクリックし、プロパティダイアログの「General Parameters」タブで「FMU file name」に該当の FMU を指定して下さい。FMU の入力・出力ポート、パラメータ情報が読み込まれます。

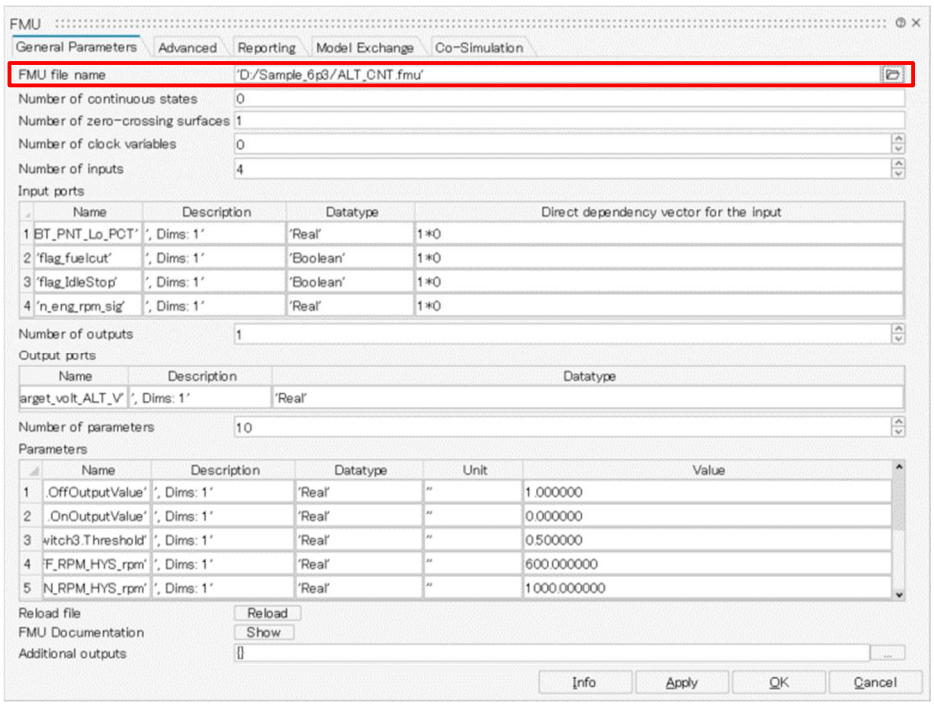


図 1.4.7　FMU のインポート

③ 「Advanced」タブでは様々な追加の挙動を指定できます。もし、Model Exchange と Co-Simulation を両方含むハイブリッド FMU を取り込む場合、このタブで「Run as Model Exchange If both FMU types are provided」にチェックを入れると Model Exchange として実行が可能です。Co-Simulation の FMU をインポートする場合は「Co-Simulation」タブで「Preferred fixed communication stepsize」で、Co-Simulation を行う際の通信間隔を設定して下さい。ダイアログで「OK」を押下すると、モデリングウィンドウ上に該当の FMU がブロックとして配置されます。

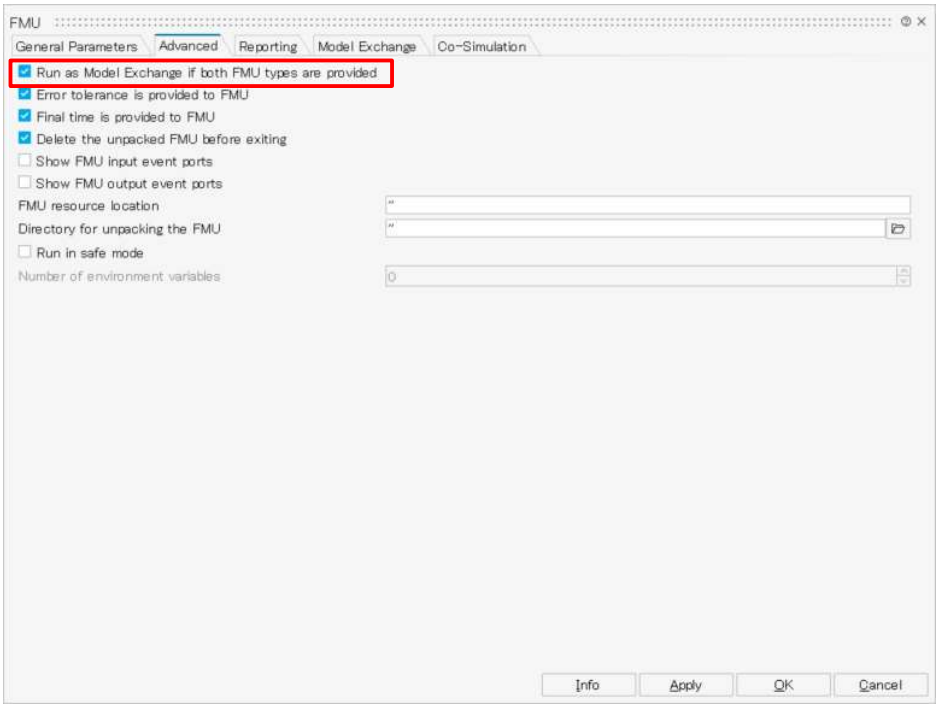


図 1.4.8　Advanced タブ

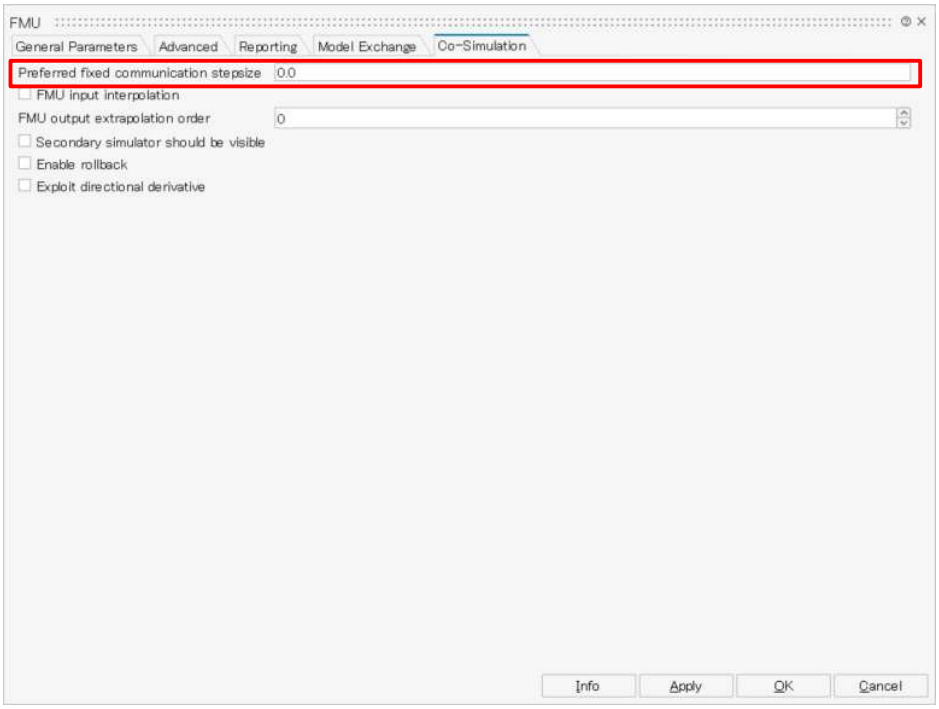


図 1.4.9　Co-Simulation タブ

④　必要な FMU を上記のように全てインポートし、FMU ブロックのポートには信号名が表示されますので適切に結線して下さい。下記、（図 1.4.10）は複数の FMU を含むモデルの例です。

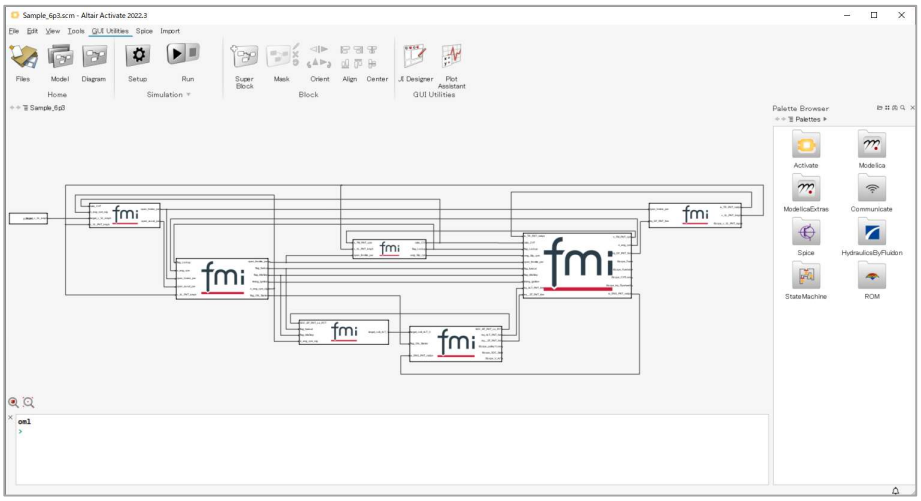


図 1.4.10　FMU 接続モデル

⑤　上部リボンの「Simulation」の「Setup」から実行時間やソルバなどのシミュレーションパラメータを設定し、同「Simulation」の「Run」ボタンからシミュレーション実行を行って下さい。
グラフでシミュレーション結果を見たい場合は、パレッドブラウザ内 Activate ライブリの「Signal Viewers」から「Scope」ブロックなどをモデルに接続して下さい。

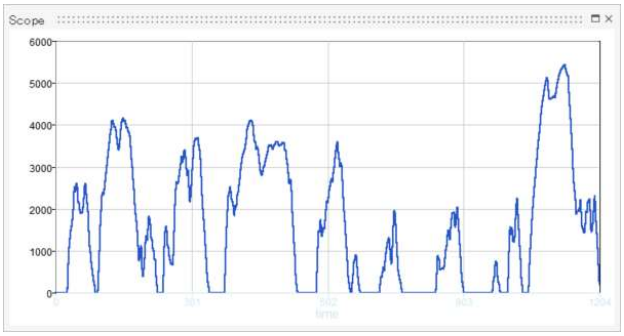


図 1.4.11　シミュレーション実行結果

1.5. dSPACE　SystemDesk および VEOS

dSPACE では FMI2.0/3.0 に準拠した FMU 生成ツールとして SystemDesk、FMI2.0/3.0 に準拠した FMU を PC 上でシミュレーションするツールとして Virtual Ecu Offline Simulator (VEOS) をそれぞれ表 1.5.1 および表 1.5.2 の仕様で用意しています。

表 1.5.1　SystemDesk インタフェース機能

SystemDesk の FMI 作成機能	
FMI バージョン	2.0、 3.0
形式	Co-Simulation
OS	Windows, Linux
実行ライセンス	不要

表 1.5.2　VEOS インタフェース機能

VEOS の FMI 読込機能	
FMI バージョン	2.0、 3.0
形式	Co-Simulation
OS	Windows, Linux
実行ライセンス	作成側に依存

FMI に準拠した FMU をシミュレーションする為の dSPACE のソフトウェアツールチェインを図 1.5.1 に示します。SystemDesk から生成された FMI 準拠の FMU を VEOS によりビルド/シミュレーションを実行します。

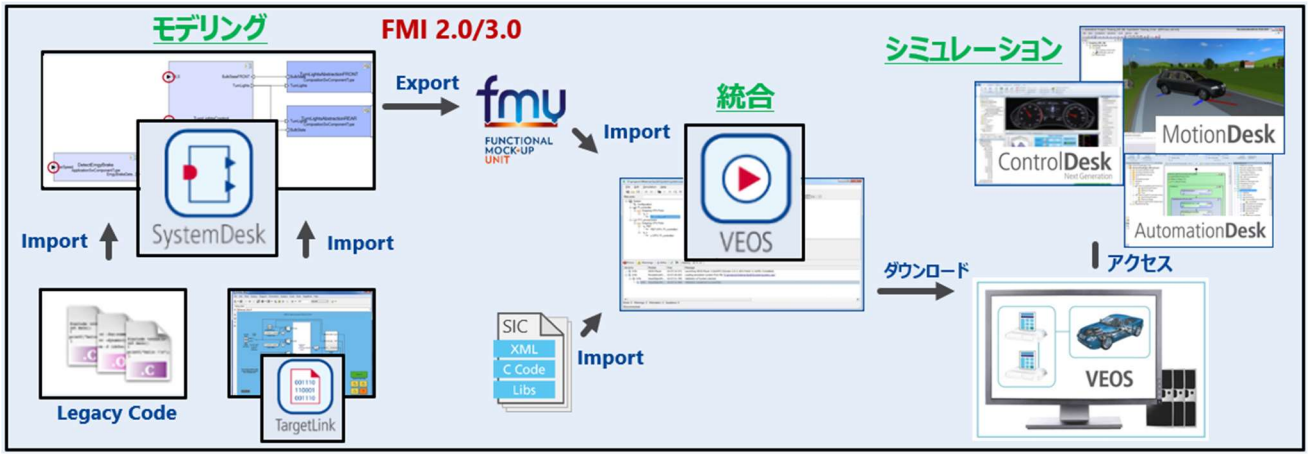


図 1.5.1 dSPACE のソフトウェアツールチェイン

1.5.1. SystemDesk による FMU 生成

SystemDesk では FMI2.0/3.0 に準拠した FMU を生成することができます。

SystemDesk は 2 つの目的を持つツールです。 1 つは AUTOSAR 準拠のソフトウェア設計ツールとして、Classic 或いは Adaptive AUTOSAR に準拠したソフトウェアコンポーネントの設計を行うことがで

きます。この設計情報を出力し、アプリケーションと結合することで AUTOSAR 準拠のソフトウェア開発を実現します。2 つ目の目的は、シミュレーション用の BSW を生成し、アプリケーションソフトウェアと組み合わせることで、AUTOSAR に準拠するソフトウェアとしてアプリケーションがどのような挙動となるか、確認することができます。この際、SystemDesk は PC 上の仮想検証を実現する為のテスト対象システムとして、バーチャル ECU(V-ECU)といわれるコンテナファイルを生成します。V-ECU は FMI 規格に準拠した形式で出力することもできます。

ここでは、FMI 規格に準拠した V-ECU を「V-ECU FMU」と定義します。

以下の手順により、SystemDesk から FMI 準拠の V-ECU FMU を生成します。
ここでは、FMI3.0 を例として手順を示します。

<前提条件>
アプリケーションソフトウェアとして、TargetLink 或いは LegacyCode から必要なファイルを SystemDesk へ取込み済。
<手順>

・ECU Configuration Manager で ECU Instance を選択し、FMU 用に新規 ECU Configuration を作成します。バス通信の有無、その種類により適切な ECU Configuration を選択し、必要な BSW モジュールを図 1.5.2 のように生成します。
ここで、FMI2.0/3.0 の選択肢がある為、FMU を生成する場合は選択する必要があります。

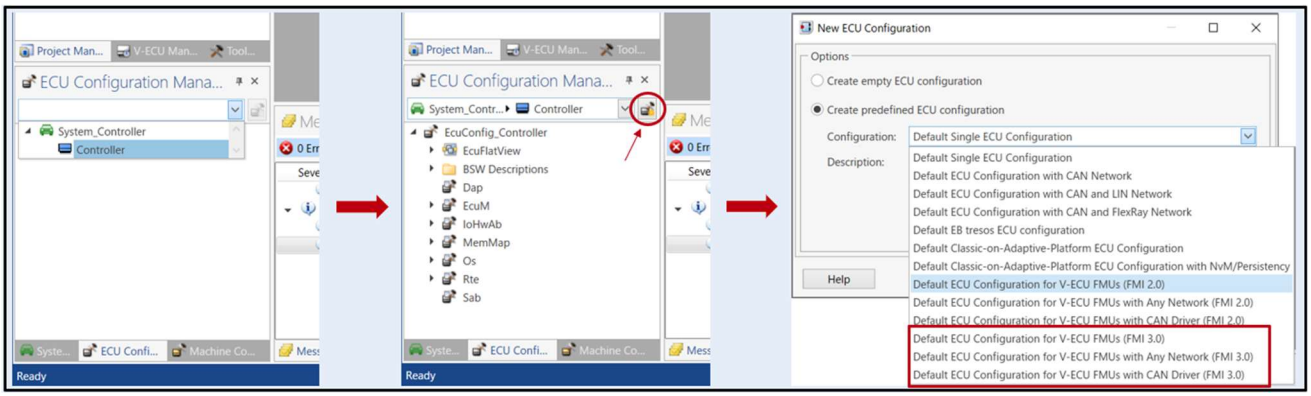


図 1.5.2　V-ECU の生成

・新規 ECU Configuration に対し、Edit Runnable Mapping および Generate Mappings を図 1.5.3 のように実行します。
ECU Configuration の最初に必要な設定となります。
アプリケーションの各ランナブルが OS の周期タスクに関連付けられます。

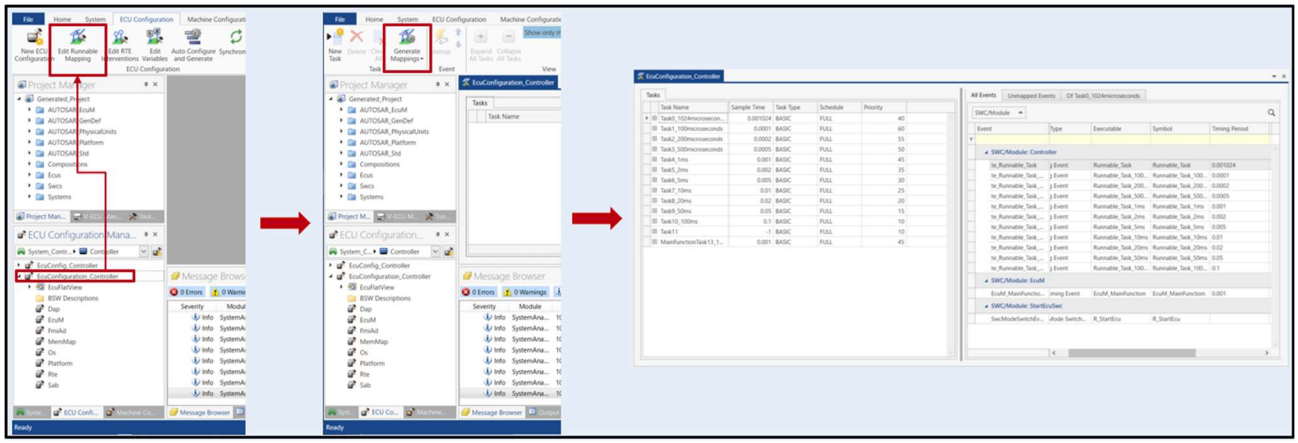


図 1.5.3 ECU Configuration

・Edit RTE Interventions、Enable Intervention Services、Create RTE Intervention Points を図 1.5.4 のようにそれぞれ実行します。

RTE インターベンションは、シミュレーションの実行時にソフトウェアコンポーネントの通信にアクセスして上書きするために、元の RTE コードに追加する設定です。
例えば、ソフトウェアコンポーネントのテストを行う場合など、スティミュラス信号の入力やエラー生成を行うことができます。

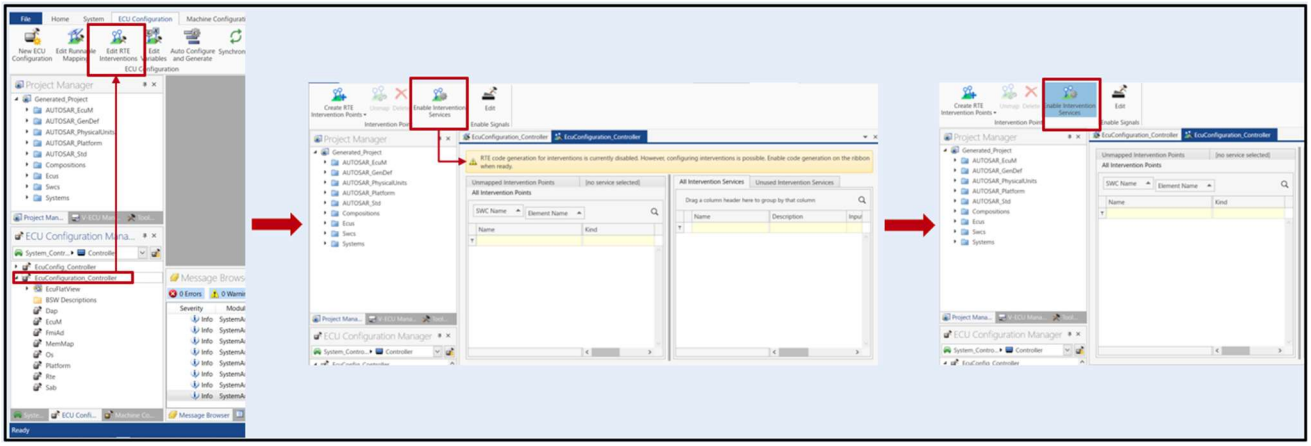


図 1.5.4 RTE の設定

・Create RTE Intervention Points のプルダウンから …and Services (Data Access Point) を図 1.5.5 のように実行します。
外部との I/F 信号を生成します。　VEOS(シミュレーションプラットフォーム)へビルドした際、他のコンテナファイルと信号を接続する為の I/F となります。

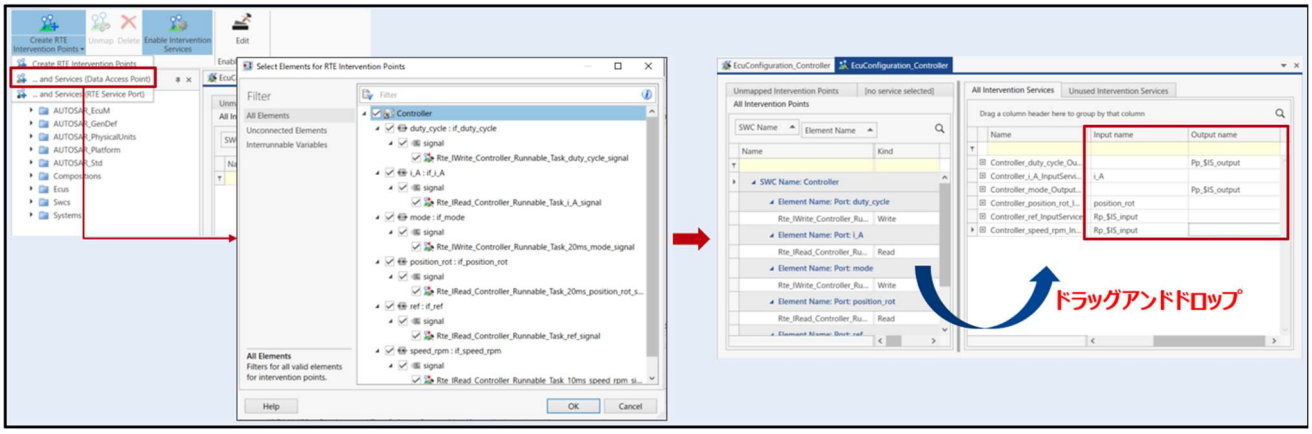


図 1.5.5 RTE の生成

・Auto Configure and Generate を実行します。
ECU Configuration で設定された内容に従い、BSW の C コードが図 1.5.6 のように生成されます。

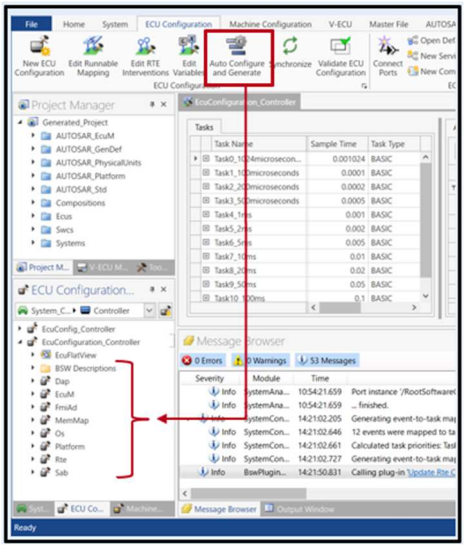


図 1.5.6 BSW の C コード生成

・V-ECU Manager から FMU コンテナを図 1.5.7 のように作成します。
V-ECU コンテナファイルに必要な C コード、a2l ファイルなどを階層化し格納します。

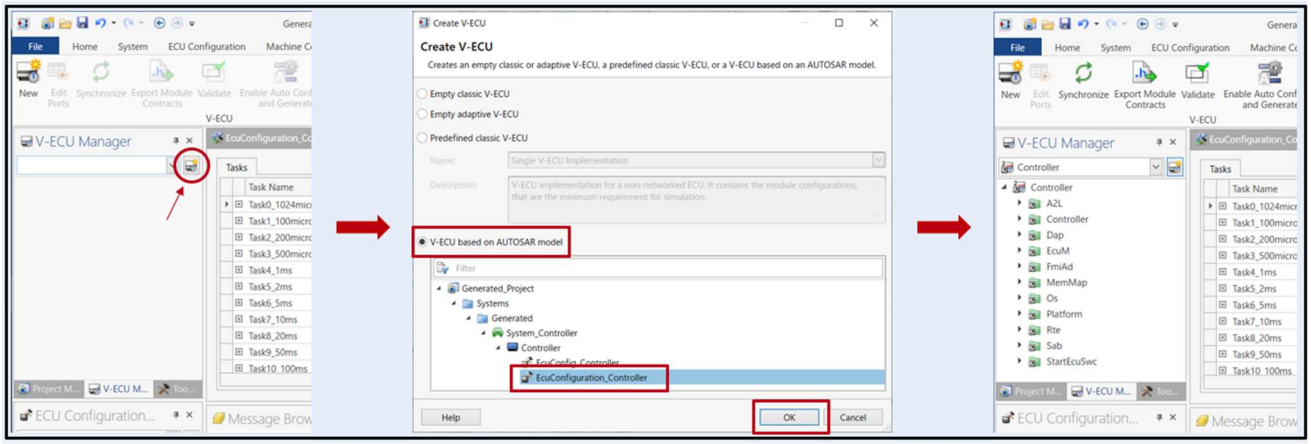


図 1.5.7 FMU コンテナ作成

・最後に、作成した FMU コンテナを図 1.5.8 のように Export します。(V-ECU FMU の生成)
Windows/Linux、 GCC/MSVC、 32bit/64bit など、V-ECU FMU を用いてシミュレーションを行うプラットフォームの環境を選択します。

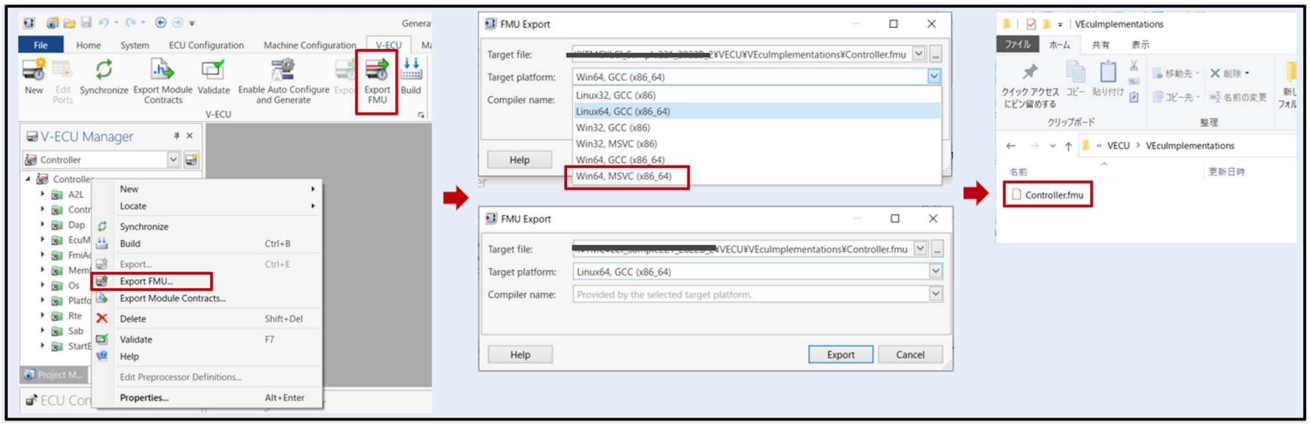


図 1.5.8 V-ECU FMU 生成

1.5.2. VEOS による FMU シミュレーション

FMI2.0/3.0 に準拠した FMU は VEOS でシミュレーションを実行することができます。
VEOS は PC ベースのシミュレーションプラットフォームであり、実機の車両などがない開発の早期の段階においても、制御モデルから生成したバーチャル ECU(V-ECU)、ネットワークを含むバスシステム、および車両モデルに至るまで、多種多様なモデルをシミュレートすることができます。

以下の手順により、VEOS で FMU をビルドし、シミュレーションを実行します。

<前提条件>
SystemDesk などの生成ツールを用いて FMI 準拠の FMU が生成されている。

<手順>
・Import から、VEOS へ実装する FMU を図 1.5.9 のように選択します。

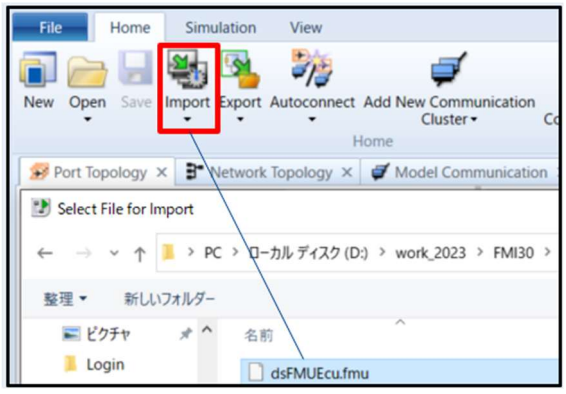


図 1.5.9 FMU の選択

・VEOS でビルドするコンパイラを図 1.5.10 のように選択します。
選択するコンパイラは、FMU 生成時に使用したコンパイラと同じものを選択してください。
(FMU 内のライブラリ作成に使用されたコンパイラ)

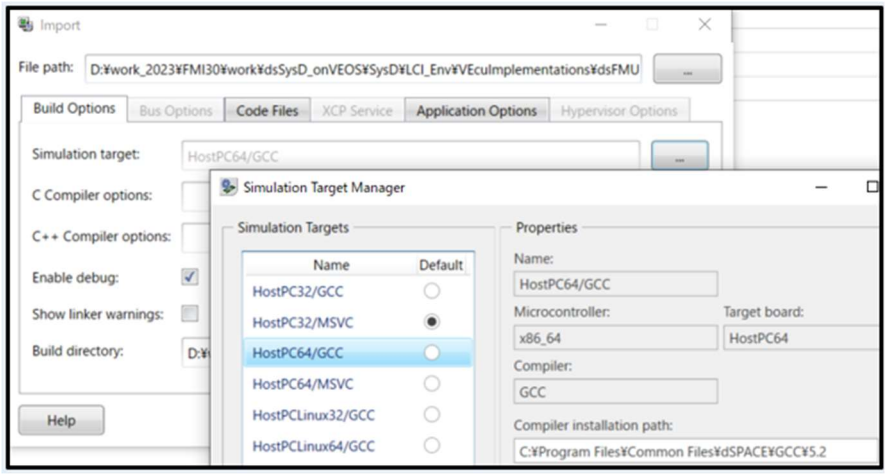


図 1.5.10 コンパイラの選択

・Build を実行すると、図 1.5.11 のように FMU が VEOS でビルドされ、Build 時にエラーが発生しなければ、VEOS 上に FMU が取り込まれたことが表示されます。

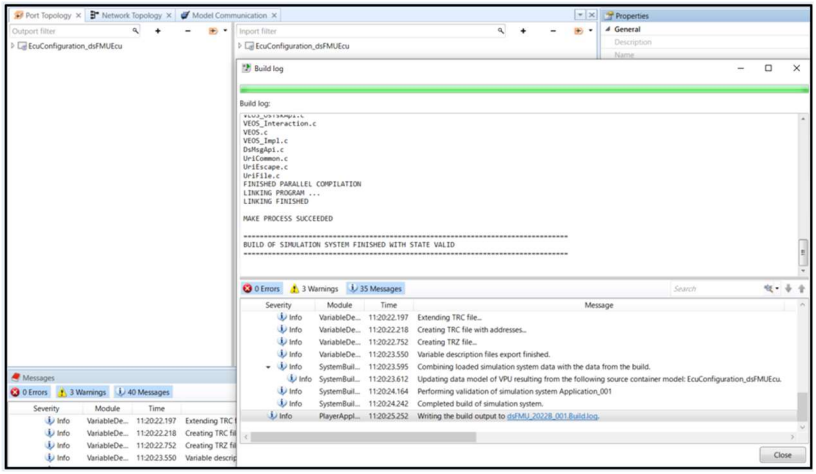


図 1.5.11 FMU のビルド

・Start ボタンを押すことにより、図 1.5.12 のように VEOS プラットフォーム上でシミュレーションが実行されます。

ControlDesk 等の計測用ツールに VEOS を Platform として登録することで、VEOS 上で動作する FMU の挙動を確認することができます。

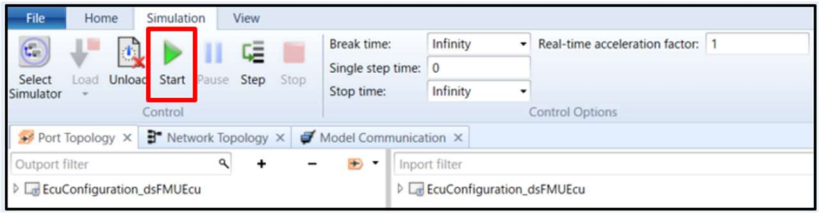


図 1.5.12 シミュレーションスタート

1.5.3. VEOS による FMI3.0 を活用したシミュレーション

FMI3.0 に準拠した FMU を VEOS へ実装し、シミュレーションを実行する場合、以下のような実装、及びシミュレーションを実現することで FMI3.0 の仕様を効果的に活用することができます。

1) 配列や整数データ型の使用

FMI2.0 では、データ型は Real、 Integer、 String、 Boolean のみの対応、且つ配列を使用するケースでは、VEOS へ実装した際に配列要素がスカラへ分解され実装されます。

例えば、5 要素を持つ配列が I/F である場合、VEOS ビルド後は 5 つの要素に分解され、図 1.5.13 のように実装されます。この場合、I/F を接続する際には分解された信号数だけ接続を行う必要があります。

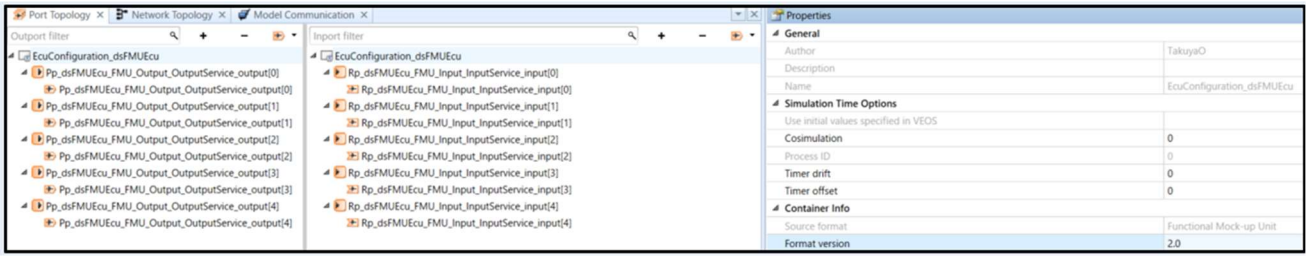


図 1.5.13 FMI2.0 準拠 FMU の実装

一方で、同様の I/F 構成を持つロジックを FMI3.0 準拠の FMU として作成した場合、配列に対応しているため、VEOS ビルド後は図 1.5.14 のように実装されます。

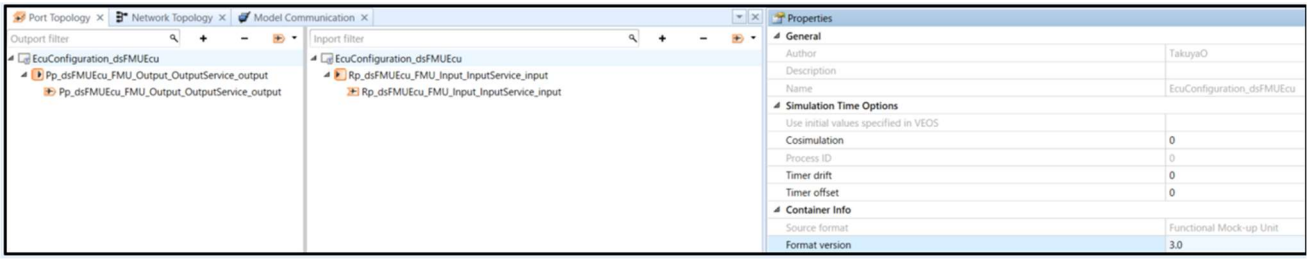


図 1.5.14 FMU3.0 準拠 FMU の実装

2) Binary I/F と Clock の使用

FMI3.0 は、新規に Binary データと Clock 信号をサポートしていますが、これらを活用し、センサデータをバス I/F でやり取りする仕組みを構築することができます。

実装の一例を図 1.5.15 に示します。2 つの FMU を CAN バスで接続し、Binary データを送受信、Clock 信号でトリガをかける実装を示しています。

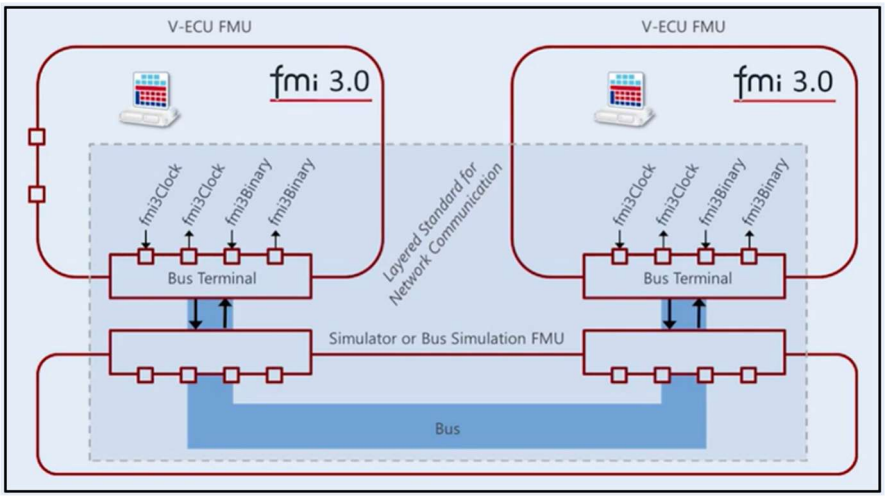


図 1.5.15 FMI3.0 の Binary 及び Clock 信号の実装

3) Binary I/F と OSI を接続したセンサシミュレーション

FMI3.0 では、例えば ASAM 標準規格 Open Simulation Interface (OSI)のような、FMU の Binary I/F と接続し、図 1.5.16 のように信号を送受信することができます。

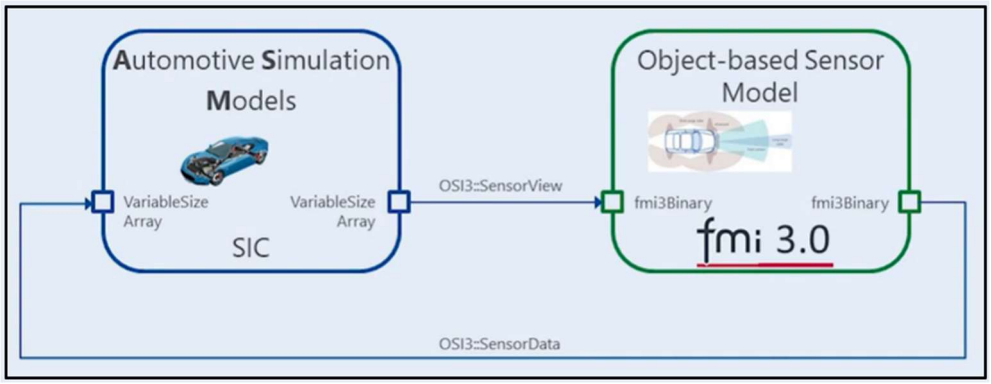


図 1.5.16 ASAM OSI を使用した Binary I/F の接続

FMI3.0 の Binary I/F を使用すると、例えば dSPACE ASM Traffic のような自車両と関連するトラフィックシナリオを含めたモデルから、SensorView 情報をセンサーモデルへ OSI を用いて送信、受信し FMI3.0 を使用したセンサーモデルが検出結果をフィードバックするような構成のシステムを実現することができます。OSI を用いたトラフィックシミュレーションの結果を図 1.5.17 に示します。

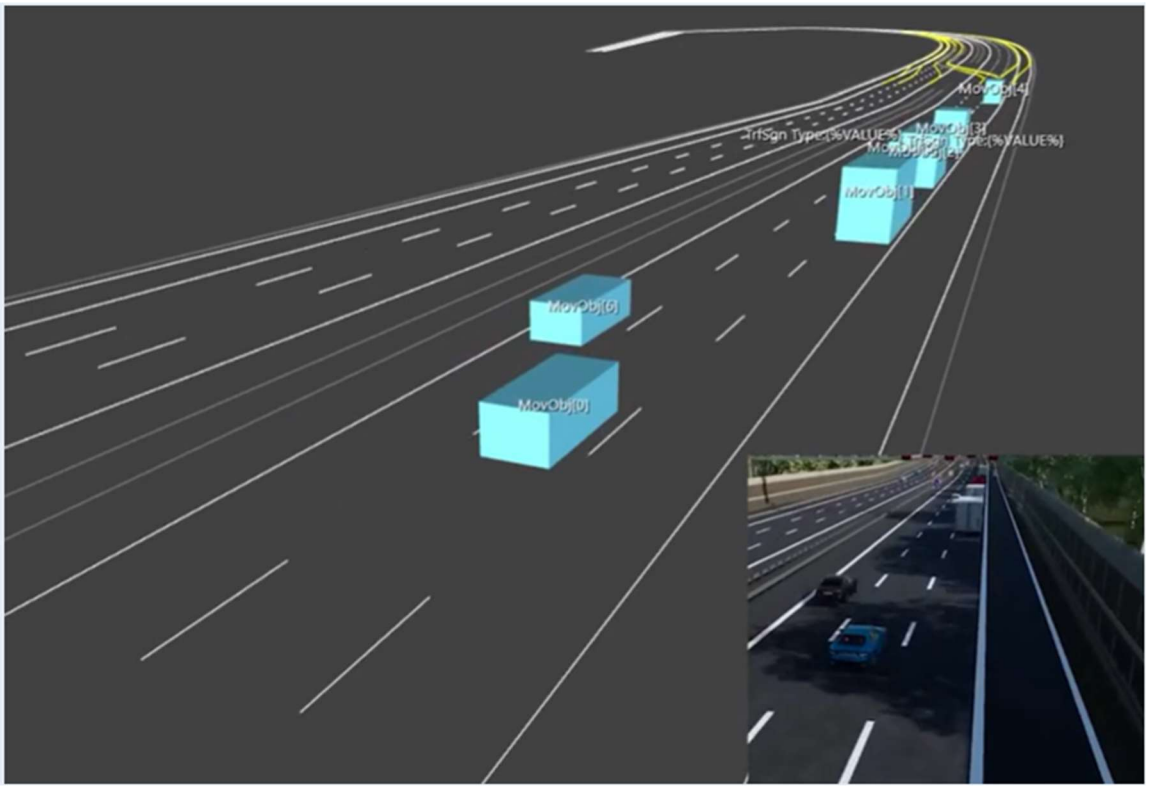


図 1.5.17 OSI を用いたトラフィックシミュレーション結果

1.6. ESI SimulationX

SimulationX は、ESI の Modelica 言語を使用したシミュレーションツールです。ツールは有償です。今回は、バージョン 4.2 での紹介です。本バージョンが有する機能について、表 1.6.1 に示します。

なお、SimulationX バージョン 4.4 以降はライセンス構成が変更されています。詳細はライセンス提供元にお問い合わせください。

表 1.6.1　SimulationX FMI インタフェース機能一覧

FMI 作成機能	
FMI バージョン	1.0、 2.0
形式	Model Exchange、 Co-Simulation
OS	Windows 64、 32
ライセンス	FMU 作成時： オプションライセンス「Code Export for FMI」が必要
	作成した FMU の実行時：不要
FMI 読込機能	
FMI バージョン	1.0、 2.0
形式	Model Exchange、 Co-Simulation
OS	Windows 64
ライセンス	FMU 読込時：不要
	読み込んだ FMU の実行時：作成側に依存

1.6.1. FMU の作成

FMU の作成方法についてご紹介します。まず、オプションライセンス「Code Export for FMI」を有効にします。その上で、メニューSIMULATION>Export > Code Export Wizard を選択します。

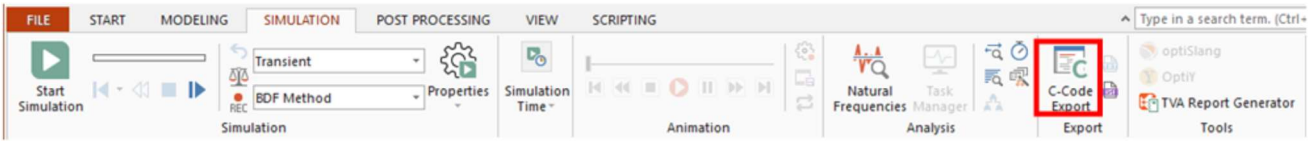


図 1.6.1 Code Export Wizard へアクセス

Code Export Wizard > Project にて、"FMI for Model Exchange"または"FMI for Co-simulation"を選択します。次に Project Name と Project Path（保存先フォルダ）を指定します。

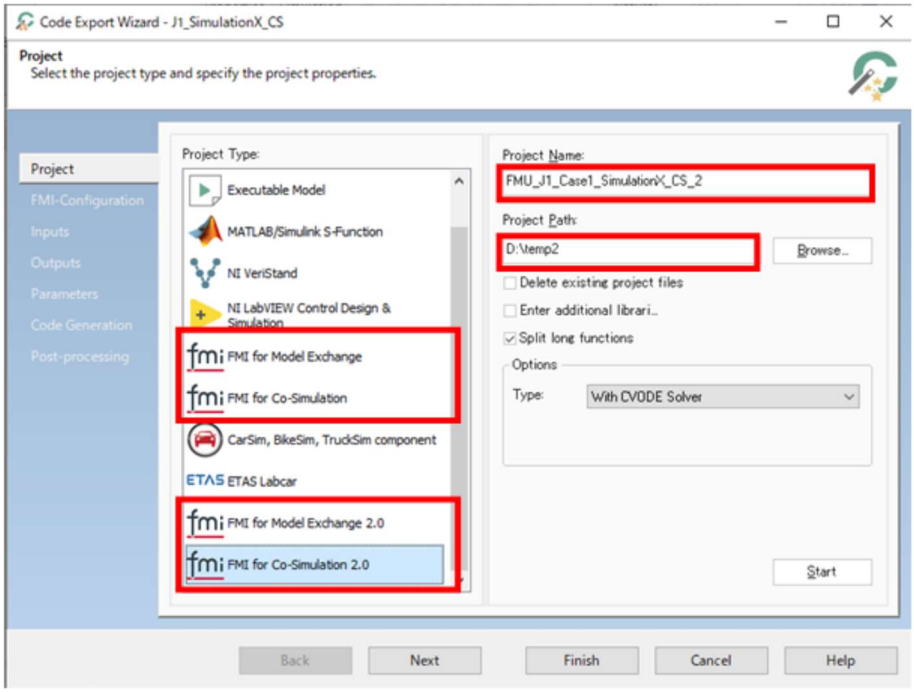


図 1.6.2 Code Export Wizard の Project

FMI-Configuration にて、FMU ブロックの作成者・バージョン・画像・解説等の設定を行います。Platform Support では作成した.fmu ファイルの DLL を使用するツールの bit 数を指定します。

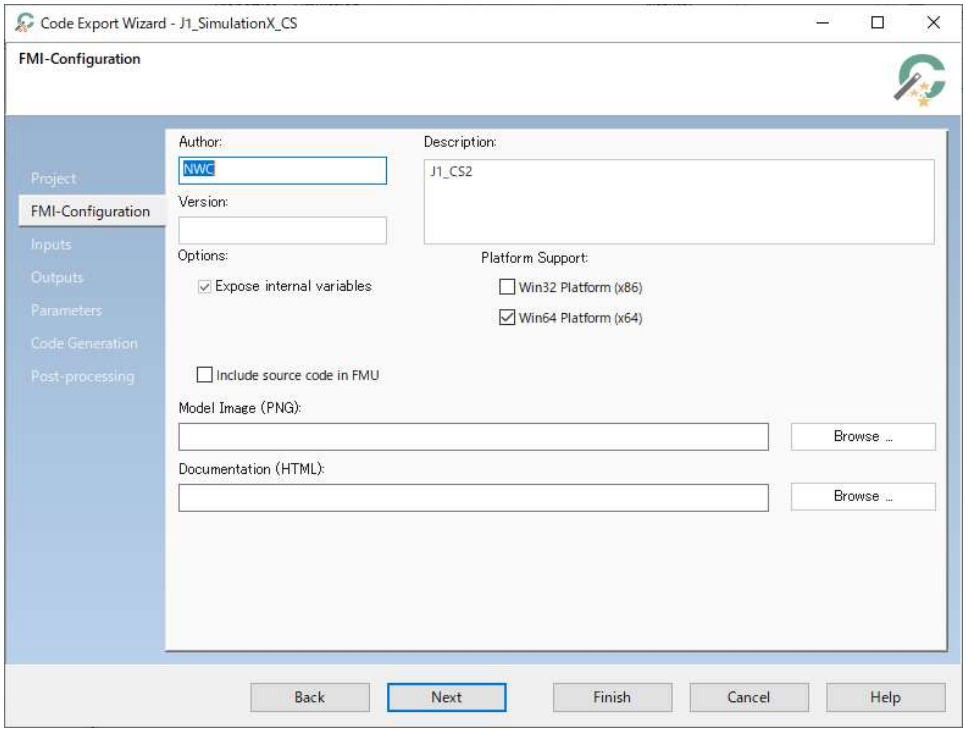


図 1.6.3 Code Export Wizard の FMI-Configuration

Inputs にて、FMU ブロックに入力するポートを選択します。対象のポートをダブルクリックもしくは Drag & Drop で"Selection"内に追加していきます。選択できるポートは Signal Input のみです。

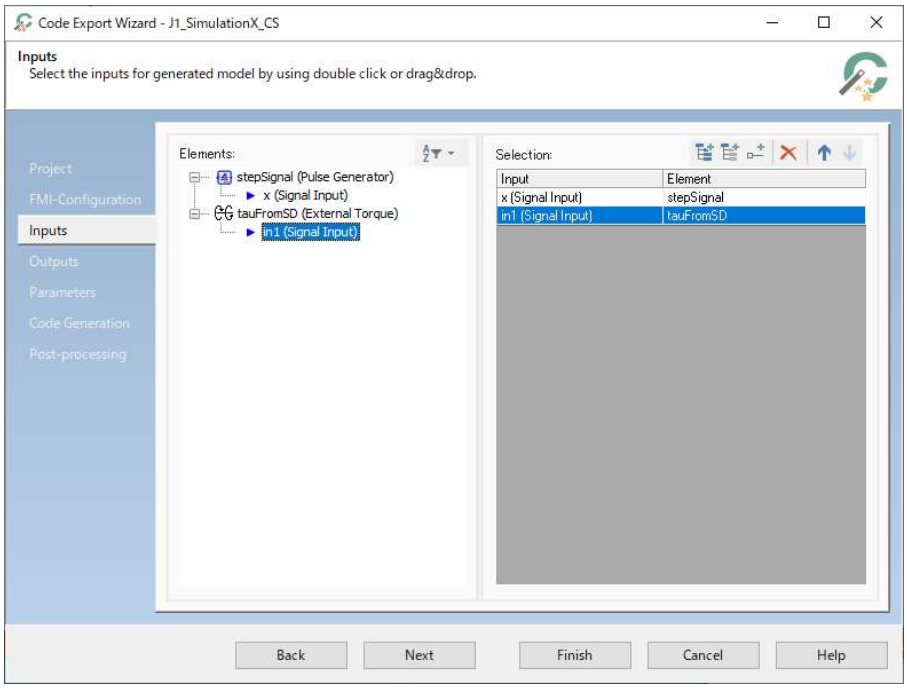


図 1.6.4 Code Export Wizard の Inputs

Outputs にて、FMU ブロックから出力する変数を選択します。対象の変数をダブルクリックもしくは Drag & Drop で"Selection"内に追加します。

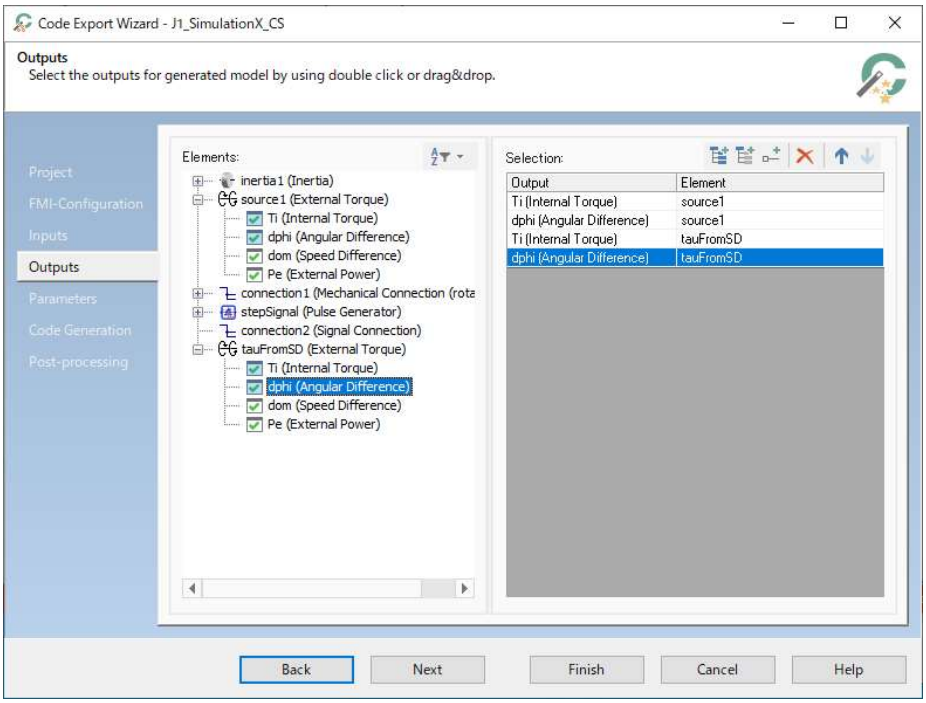


図 1.6.5 Code Export Wizard の Outputs

Parameters にて、FMU ブロックのパラメータを選択します。対象の入力変数をダブルクリックもしくは Drag & Drop で"Selection"内に追加します。

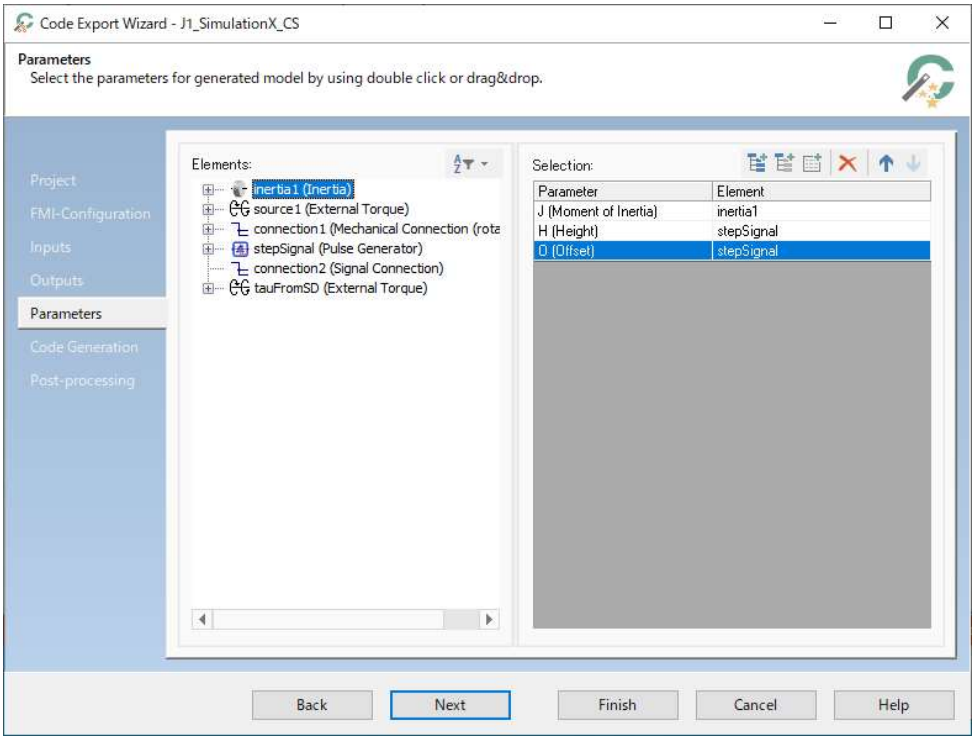


図 1.6.6 Code Export Wizard の Parameters

Code Generation に移るとモデルのソールファイルが生成されます。生成が終了すると、Project Path で指定したフォルダにソースファイル等が並びます。

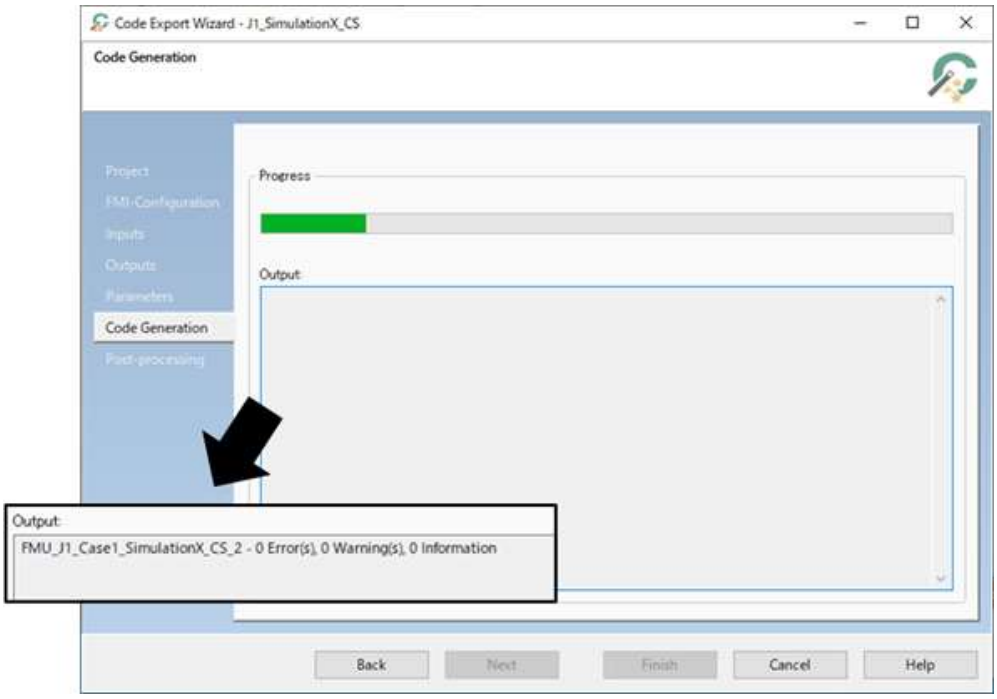


図 1.6.7 Code Export Wizard の Code Generation

Post-processing にてコンパイルを行います。コンパイラを選択した後、Build ボタンを押すとコンパイルが開始されます。保存先フォルダに FMU が作成されています。

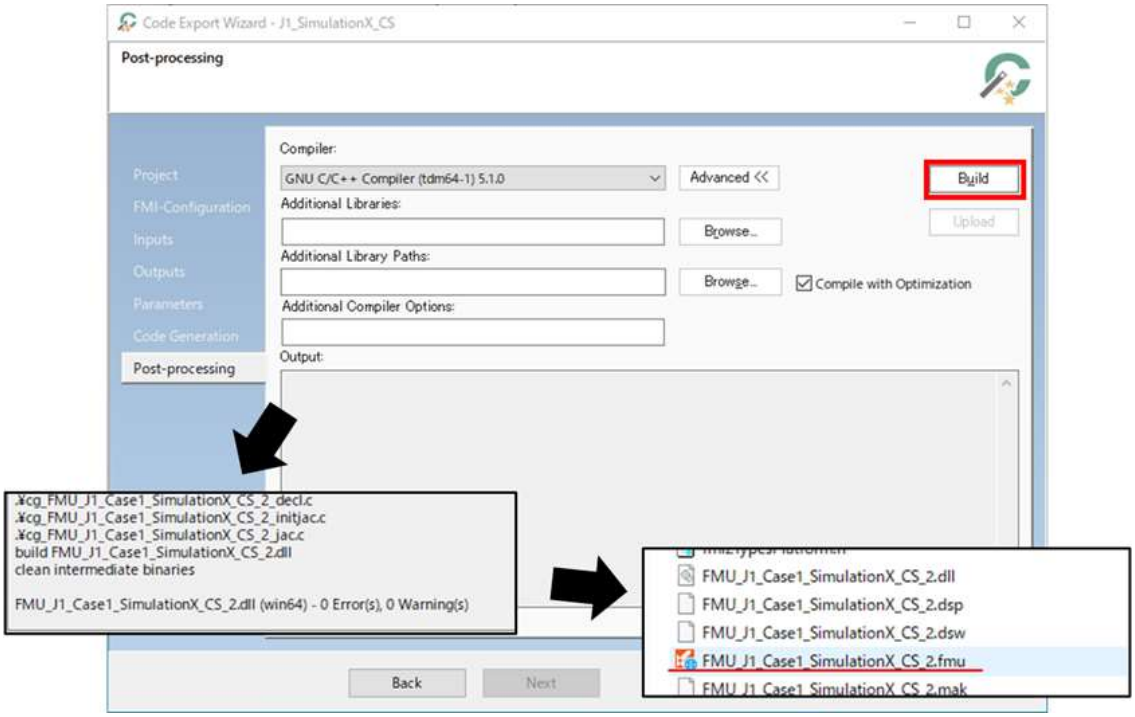


図 1.6.8 Code Export Wizard の Post-processing

1.6.2. FMU のインポート

標準機能としてインポート機能が付属されており、オプションライセンス不要で利用できます。まず、MODELING > Add FMU Block... を選択します。

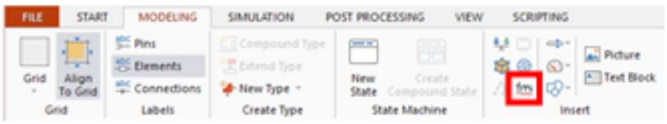


図 1.6.9 fmu インポート機能へアクセス

次に、取り込みを行う.fmu ファイルを指定します。

図 1.6.10 取り込みを行う.fmu ファイルの指定

ファイル指定後に、取り込み時のオプションを選択します。

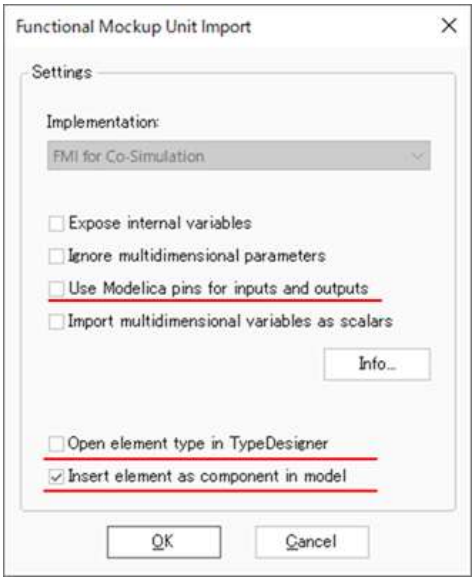


図 1.6.11 Functional Mockup Unit Import の Settings

オプションのチェックあり/なしについて以下に説明します。

「Use Modelica Pins for Inputs and Outputs」:

チェック無しでは SimulationX の Signal ポートを使用します。

チェックすると、Modelica の Block ポートを使用します。

「Open Element Type in TypeDesigner」:

TypeDesigner を開き、ポート名やパラメータ名等の設定を行います。

「Insert Element as Component in Model」:

Model View 上に取り込んだ FMU エレメントを自動で配置します。

インポート作業を FMU の数だけ繰り返します。インポートしたブロックのポート間を結線します。

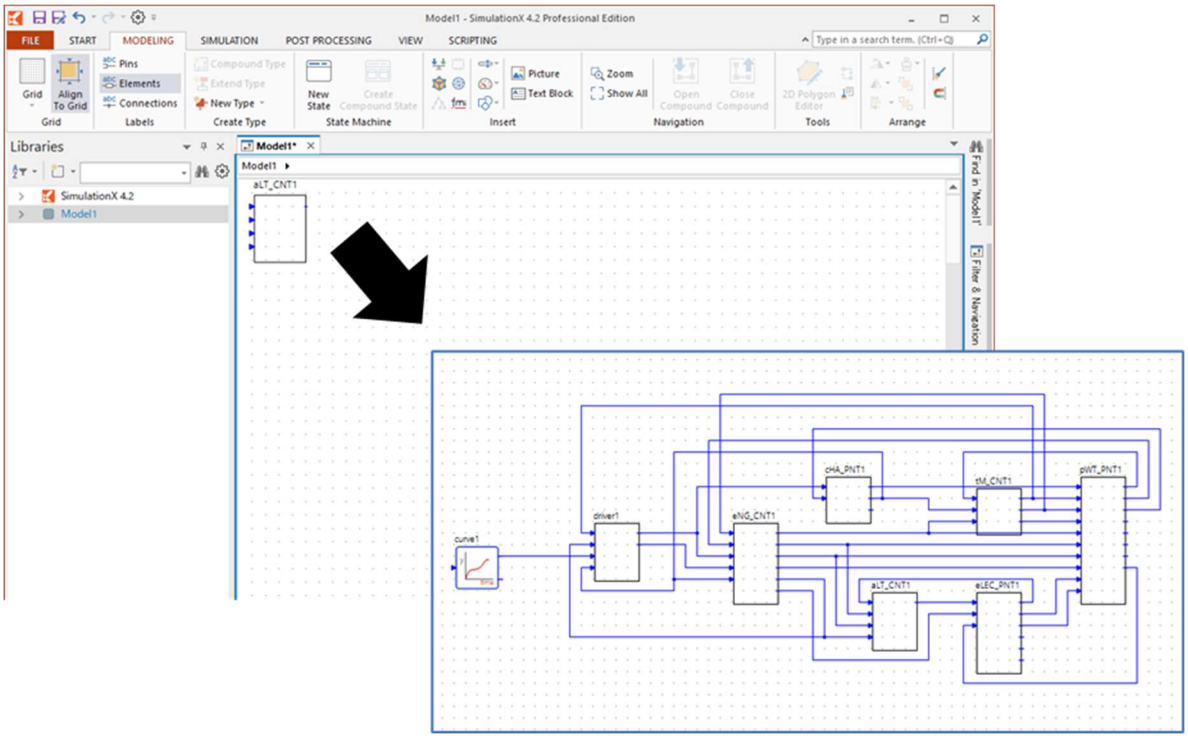


図 1.6.12 モデル接続例

1.6.3. FMU の実行

SIMULATION >Properties にて、シミュレーションの条件設定をします。

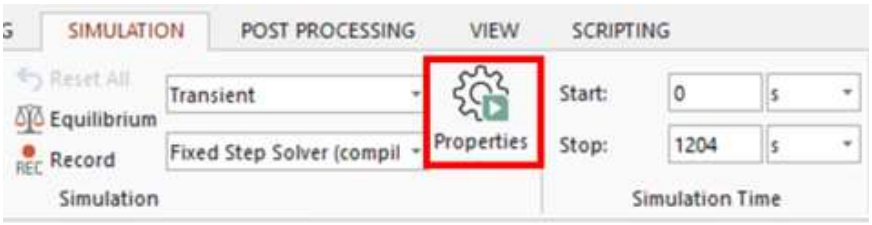


図 1.6.13 Properties へのアクセス

今回の事例では、設定する値は、以下です。

Start Time	: 0 [s]
Stop Time	: 1204 [s]
Solver, Step Sizes and Tolerances	: Fixed step Solver（固定ステップ）
Integration method	: Euler Forward （ODE1 相当）
Min. Calculation Step Size	: 0.0025 [s]
Min. Output Step Size	: 0.01 [s] ・・・出力結果のサンプル時間なので任意

Co-Simulation では FMU 毎に入力信号を受け取る時間間隔（Communication Step Size）の設定が必要です。モデルをダブルクリックして Properties> Master Algorithm にアクセスします。今回は全モデルの Communication Step を 0.0025 [s]に設定します。

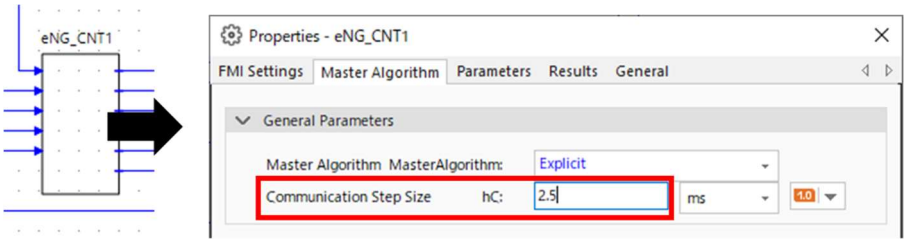


図 1.6.14 Properties > Master Algorithm> Communication Step Size

出力結果をシミュレーション中に観測するために設定が必要です。モデルをダブルクリックして Properties> Results にアクセスします。観測を有効にするためにはチェックを入れます。

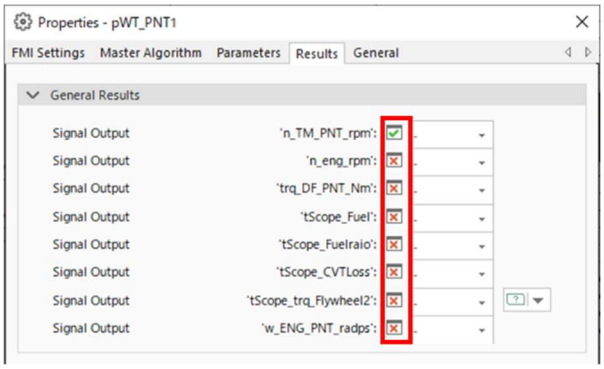


図 1.6.15 Properties > Results

SIMULATION >Start Simulation にて、シミュレーションを実行します。

図 1.6.16 Start Simulation

各インポートした FMU を右クリック、表示した Results にて出力結果を表示します。

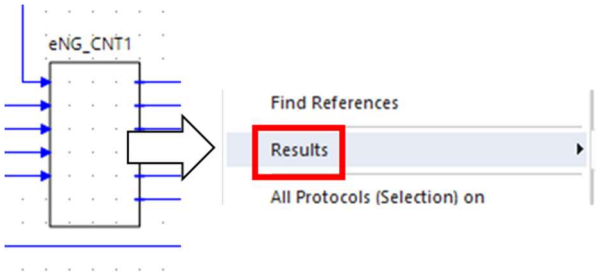


図 1.6.17 出力結果

1.7　Gamma Technologies　GT-SUITE

GT-SUITE は、米国 Gamma Technologies 社で開発されたマルチフィジックス・システムシミュレーションツールで、コンピュータ上で自動車をはじめとする様々な工業製品を設計し、任意の運転条件におけるエネルギ効率、騒音等の NV 評価、冷却系システムの熱マネージメント評価などをおこなうことができます。

FMI ついては、FMI 1.0、2.0 に対応しています。（3.0 は今後対応予定です。）

また GT-SUITE へのインポートに関しては Co-Simulation と Model Exchange に対応しています。GT-SUITE からのエクスポート（FMU 生成）に関しては、Co-Simulation の対応となります。

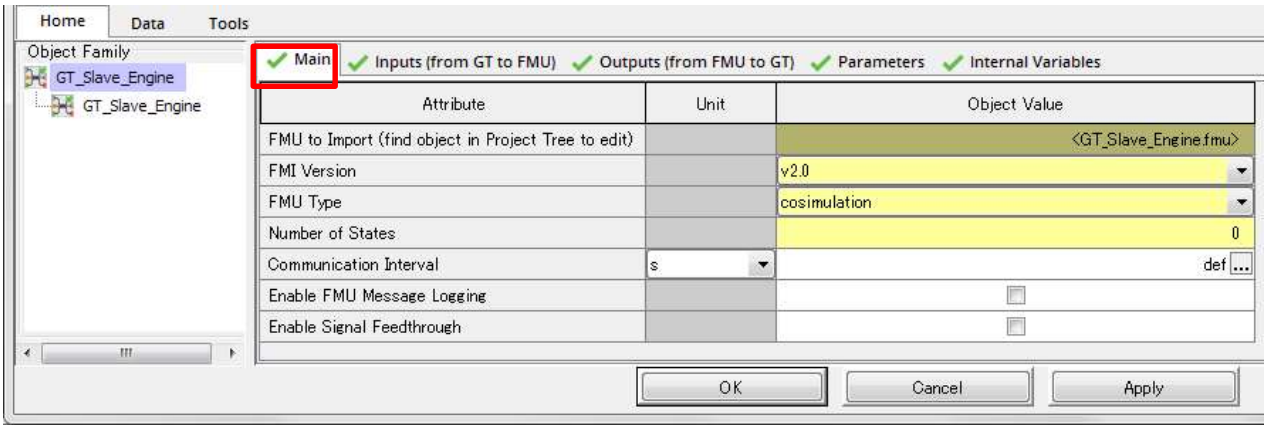
1.7.1　GT-SUITE へのインポート

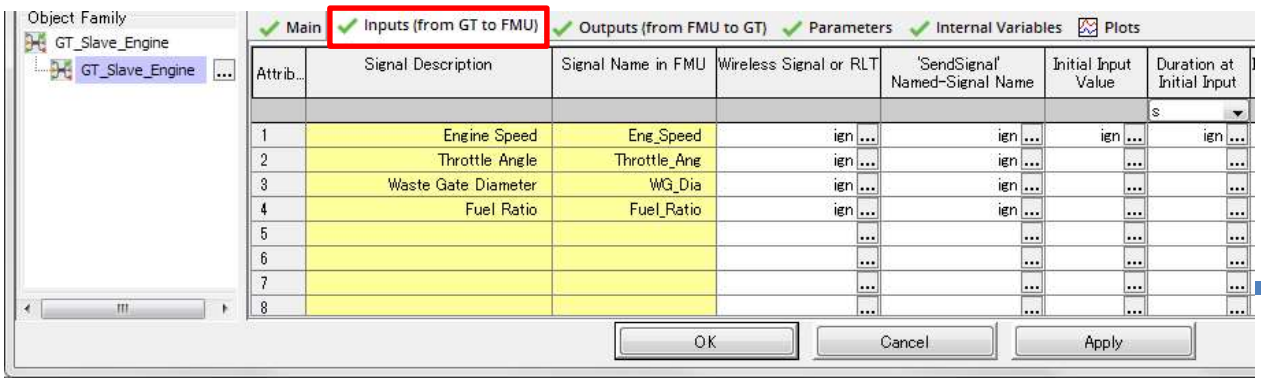
・FMUImport の設定

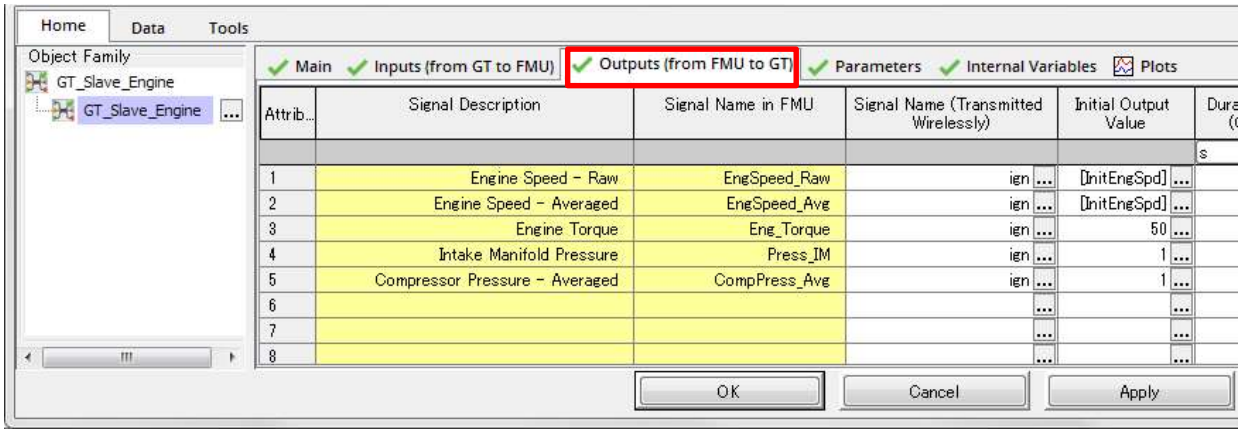
他ツールで生成された FMU を読み込むために、FMUImport オブジェクトを使用します。

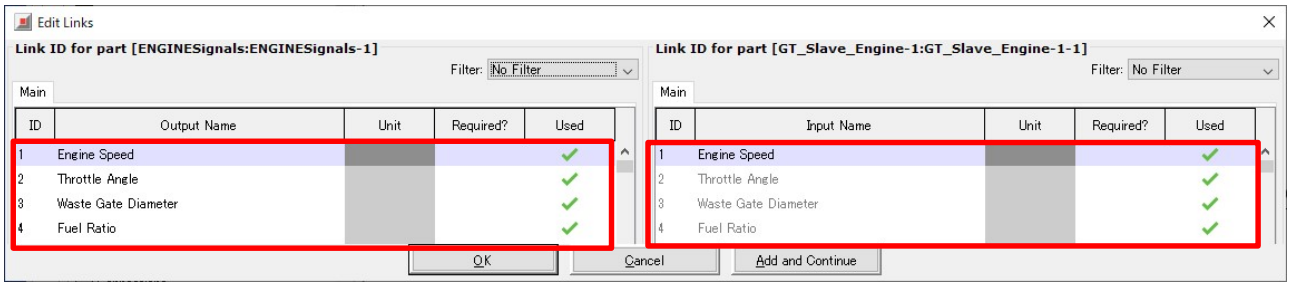
GT-ISE のライブラリから FMUImport オブジェクトを選択すると、初めに FMU（*.fmu ファイル）を選択することになります。すると Inputs/Outputs フォルダに、信号名などが読み込まれます。

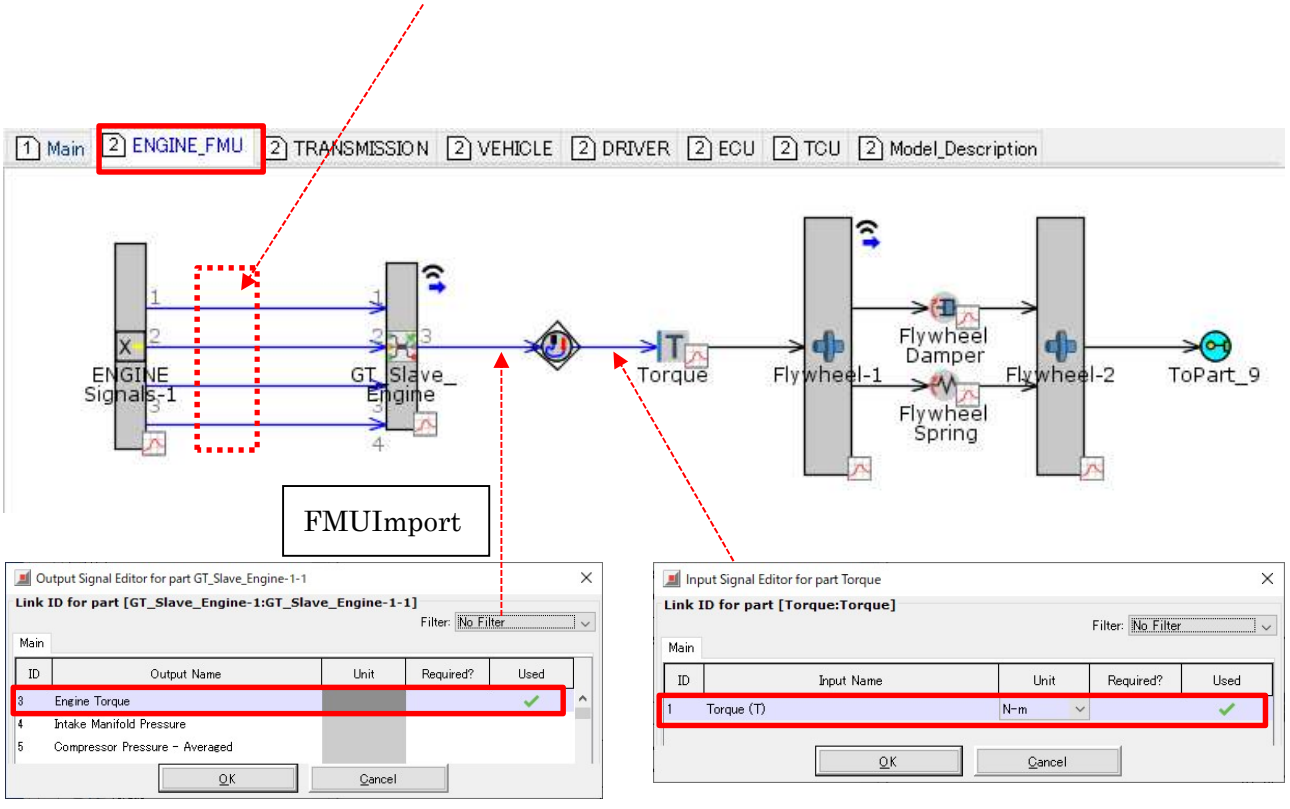
次に FMUImport をパーツとして配置します。インポートした FMU へ信号を渡すには、SensorConn を介して任意の信号を FMUImport へ接続します。（あるいは、Inputs フォルダを利用して、SendSignal から信号を受け取ったり、RLT 値を受け取ったりすることも出来ます。）FMU からの信号を受け取るには、ActuatorConn を介して FMUImport を任意のパーツへ接続します。（あるいは、Outputs フォルダを利用して、各パーツへ信号を送ることも出来ます。）入出力信号は複数個扱うことが可能です。







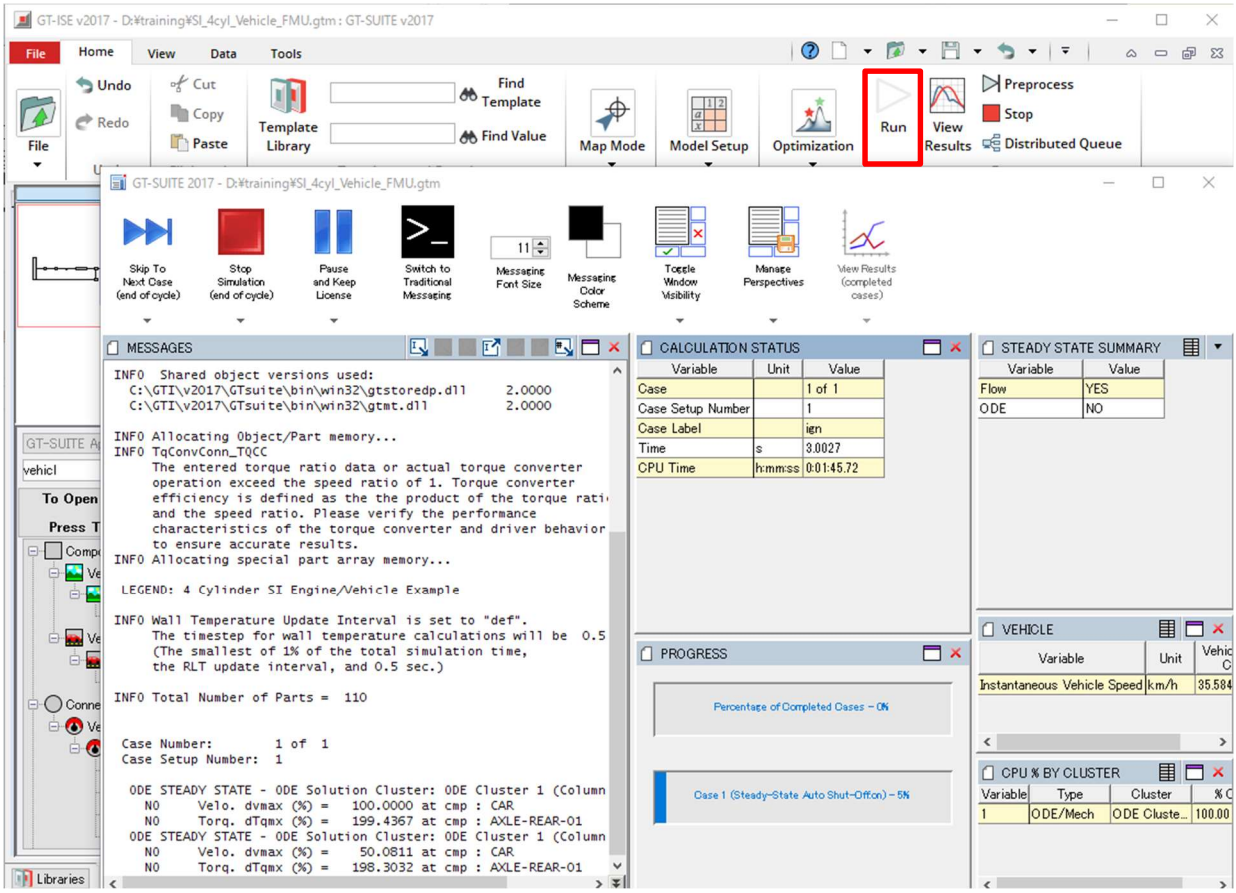




(例) FMUImport へエンジン回転数などを渡し、トルクを受け取る場

・計算実行

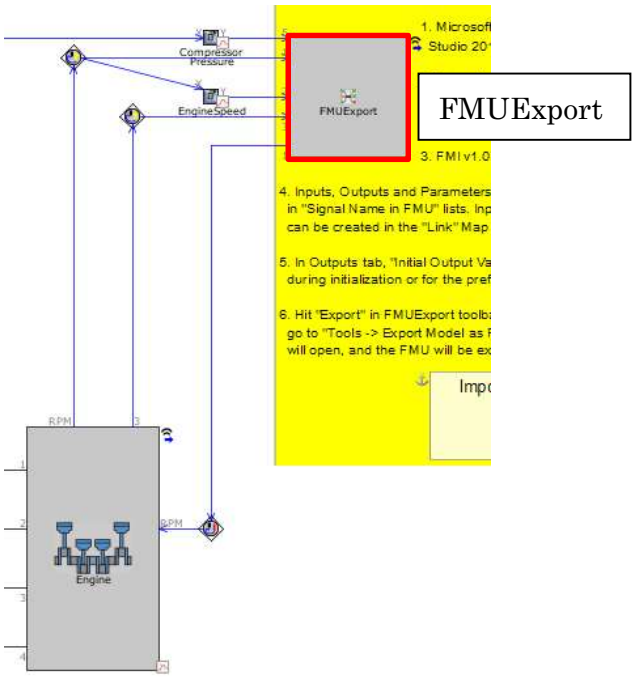
GT-ISE から通常通りシミュレーションを実行すると、計算が始まります。



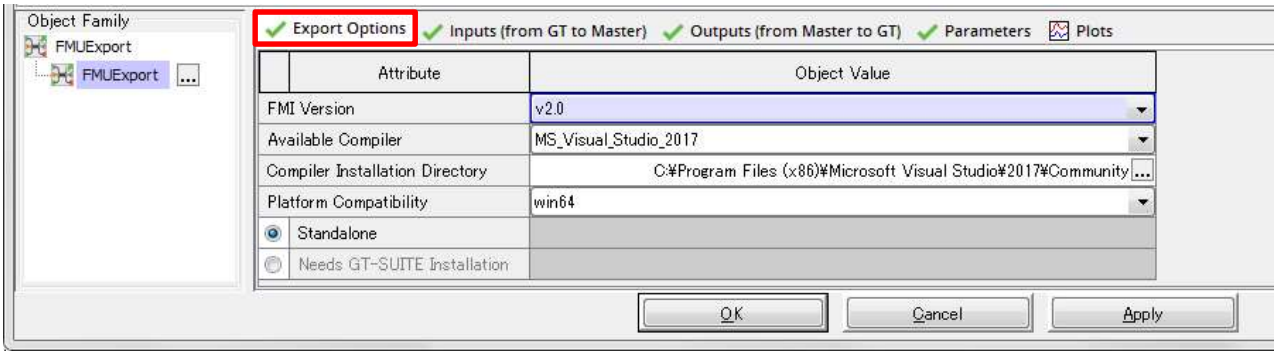
1.7.2　GT-SUITE からのエクスポート（FMU 生成）

・FMUExport の設定

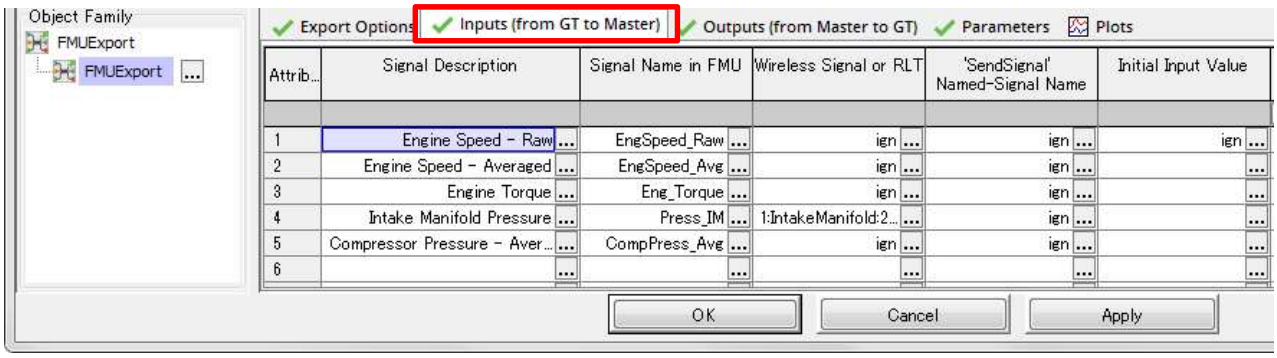
他ツールと信号の受け渡し設定をするために、FMUExport オブジェクトを使用します。

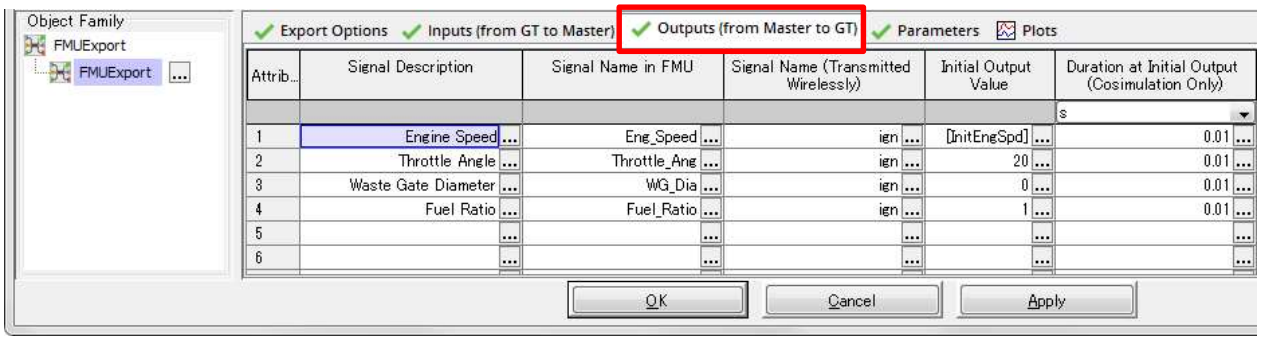


Export Option フォルダの Available Compiler では使用するコンパイラを選択し、Complier Installation Directory でインストールフォルダを指定します。Platform Compatibility では、FMU が使用される環境を指定します。ラジオボタンでは、FMU のユーザ環境において、GT-SUITE のインストールが必要とするかどうかを選択します。

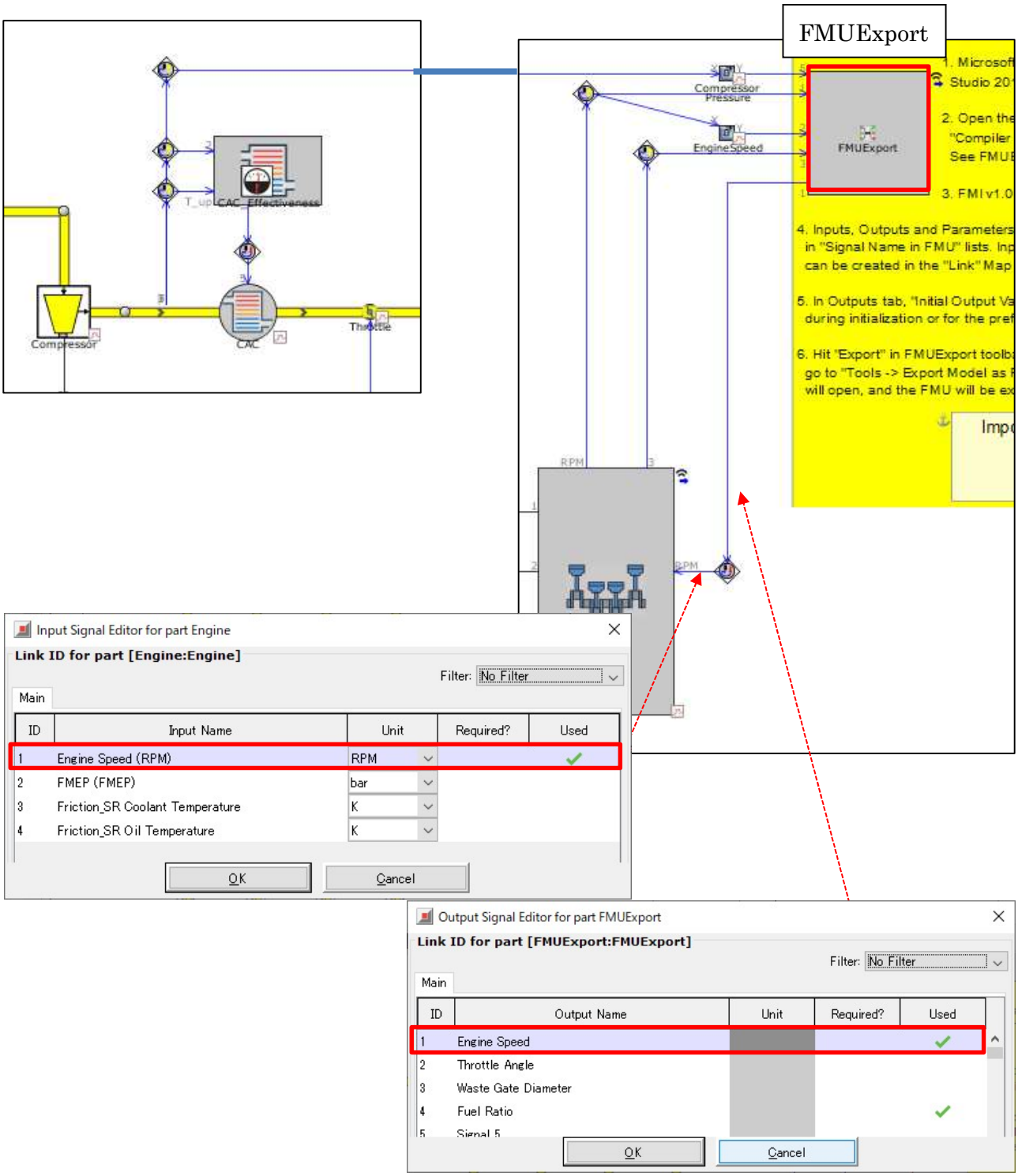


Inputs/Outputs フォルダでは、受け渡す信号を指定します。Inputs フォルダは、GT-SUITE 側から他ツールへ渡す信号を設定し、Outputs フォルダでは、他ツールから GT-SUITE 側へ渡される信号を設定します。Signal Description は単なる説明書きで、モデル計算上使用されることはありませんが、Signal Name in FMU は FMU としての信号名となり、他ツールなどで読み込まれます。

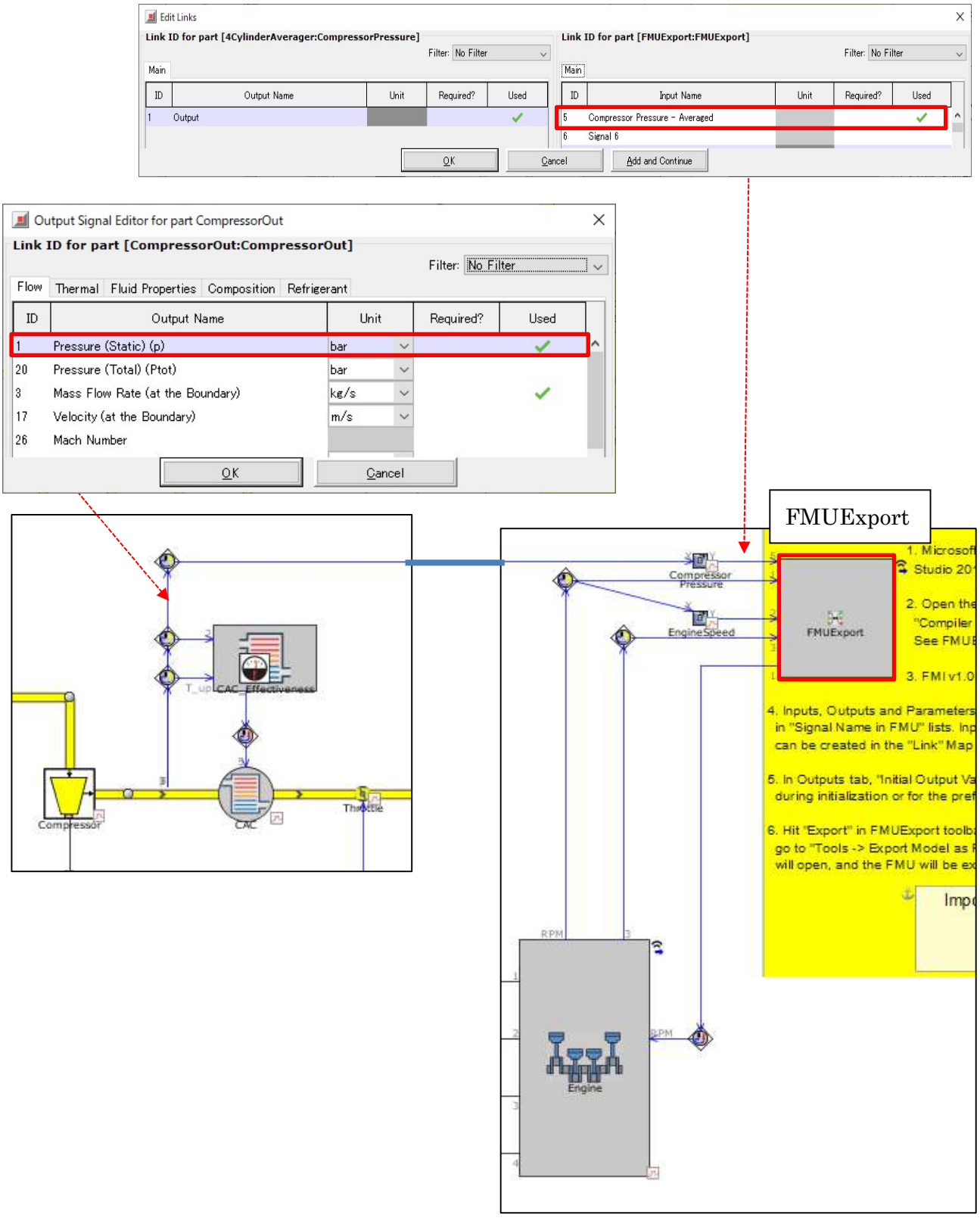




他ツールへ信号を渡すには、SensorConn を介して任意の信号を FMUExport へ接続します。（あるいは、Inputs フォルダを利用して、SendSignal から信号を受け取ったり、RLT 値を受け取ったりすることも出来ます。）他ツールからの信号を受け取るには、ActuatorConn を介して FMUExport を任意のパーツへ接続します。（あるいは、Outputs フォルダを利用して、各パーツへ信号を送ることも出来ます。）入出力信号は複数個扱うことが可能です。



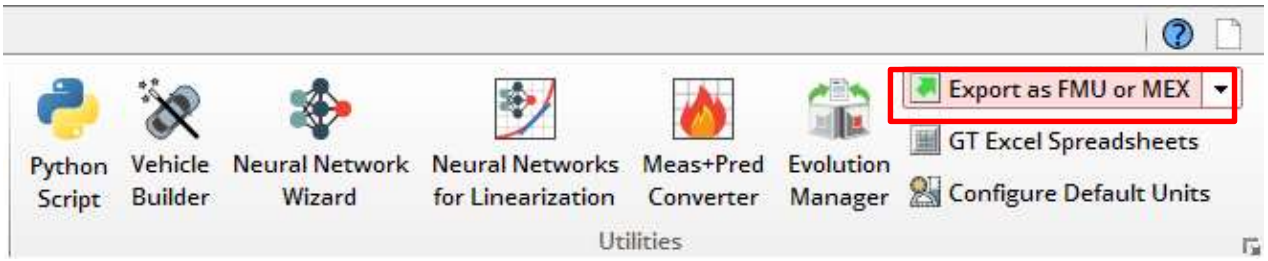
（例）他ツールからエンジン回転数を受け取る場



（例）他ツールにコンプレッサの圧力を渡す場

・エクスポート

GT-ISE のメニューの Tools リボンから Export as FMU or MEX を選択すると、自動的に FMU（*.fmu ファイル）が生成されます。



生成した FMU は、他ツールで読み込み使用することが可能です。

1.8　ETAS　VECU-BUILDER

ETAS が提供する FMU 生成ツール VECU-BUILDER は以下の機能を備えています。

- FMI 2.0 準拠
- 生成する FMU に XCP スレーブ機能を付与(単体で INCA などと連携が可能)
- 生成する FMU は Solver を内蔵しコシミュレーション可能
- C ソースコード、オブジェクトファイル、DLL からの FMU 生成
- コマンドラインによる実行
- Windows および Linux に対応

FMU の生成に特化することにより、ETAS COSYM を始めとした FMI 準拠のコシミュレータとのツールチェーンを柔軟に構成することが可能となっています。

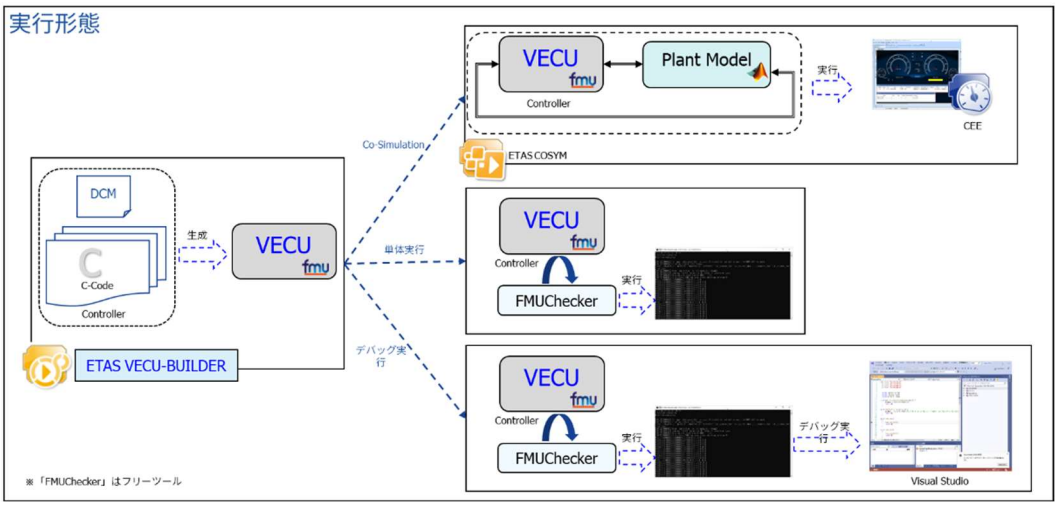


図 1.8.1VECU-BUILDER とツールチェーン構成例

また、コマンドライン・インターフェースであるため、ビルド・パイプラインのバッチ処理に用意に組み込むことができます。

1.8.1　FMU の作成

VECU-BUILDER は、FMU の元となるファイル(C ソースコード、オブジェクトファイル、DLL など)と YAML 形式の設定ファイルから FMU を生成します。

YAML ファイルでは、入力ファイル、入出力および参照する信号、タスク周期、EEPROM の動作、外部のインクルード・ファイルなどを詳細に設定することが可能です。

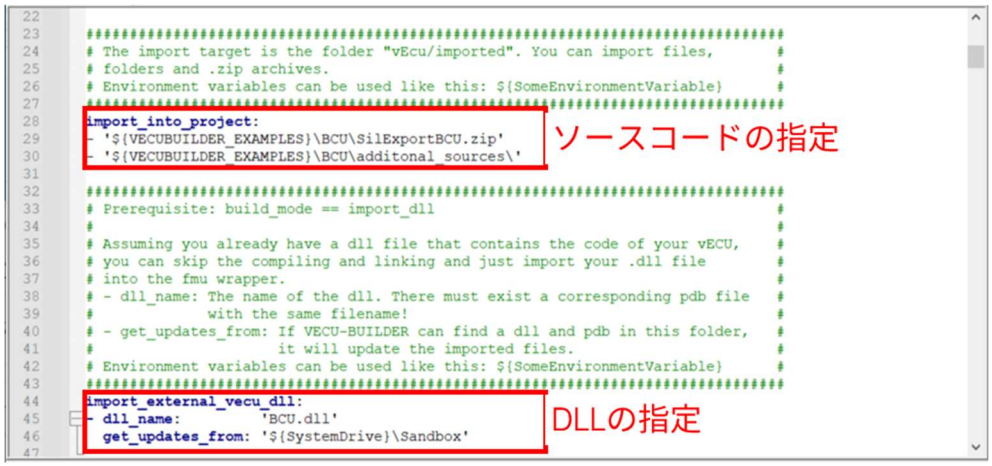


図 1.8.2 YAML ファイル例: 入力ファイルの設定

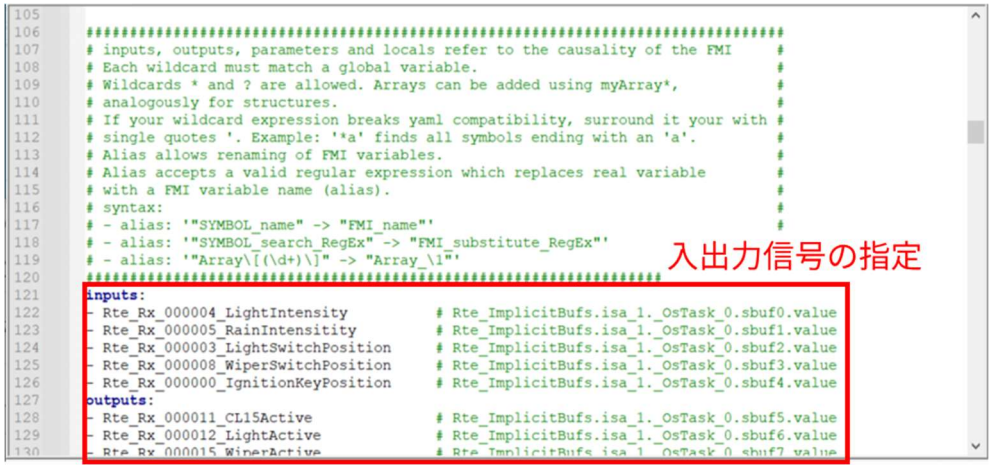


図 1.8.3 YAML ファイル例: 入出力信号の設定

FMU はコマンドライン実行により生成されます。FMU のビルドや簡易動作確認、デバッグ実行などの一連の必要な操作は、バッチファイルとして VECU-BUILDER のなかで用意されています。

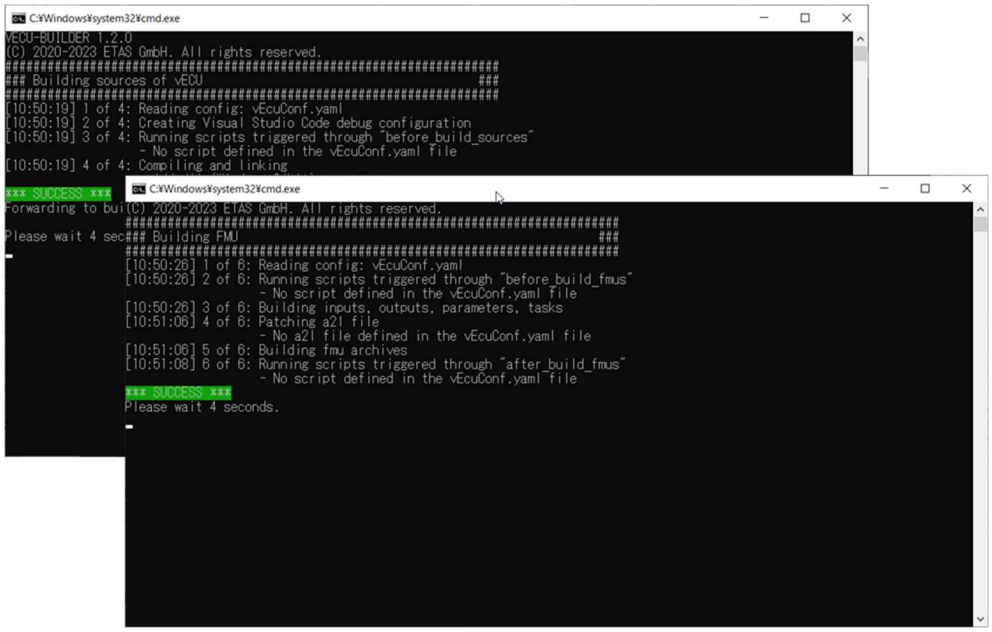


図 1.8.4 FMU ビルドの様子(コマンドライン実行)

1.8.2　FMU のインポートと実行

FMU の実行に際しては、FMI に準拠したコシミュレータを使用します。

ここでは ETAS のコシミュレータ COSYM を用いて説明します。

まず、実行したい FMU をインポートします。これはコンテキストメニューの「Import model」から行えます。

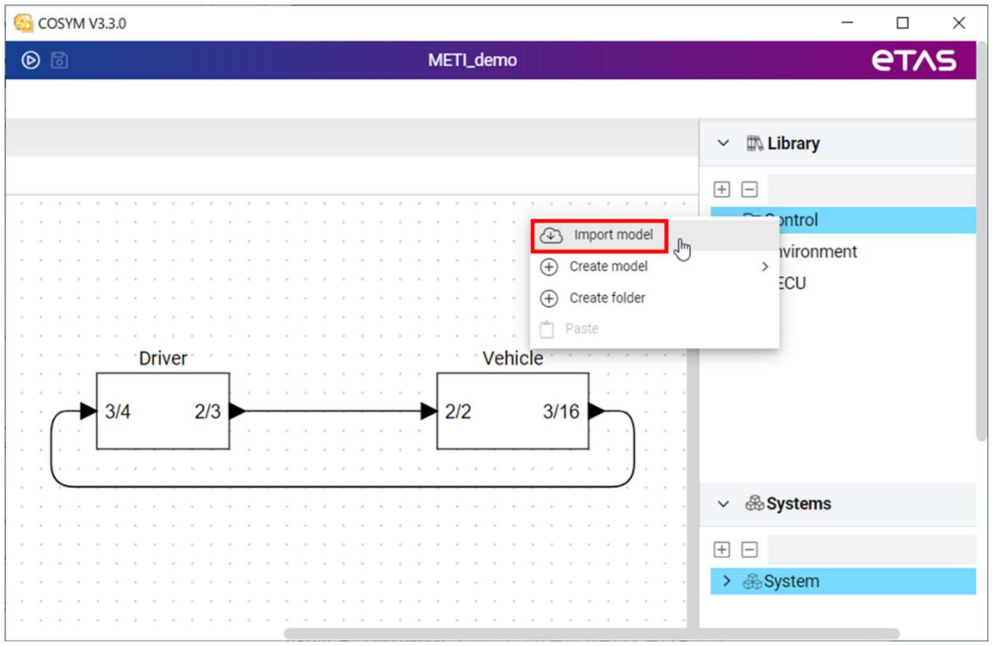


図 1.8.5 FMU のインポート

次に、インポートした FMU をシミュレーションするシステムに追加します。これはインポートしたモデルのリストから設計画面へのドラック・アンド・ドロップにより実行できます。

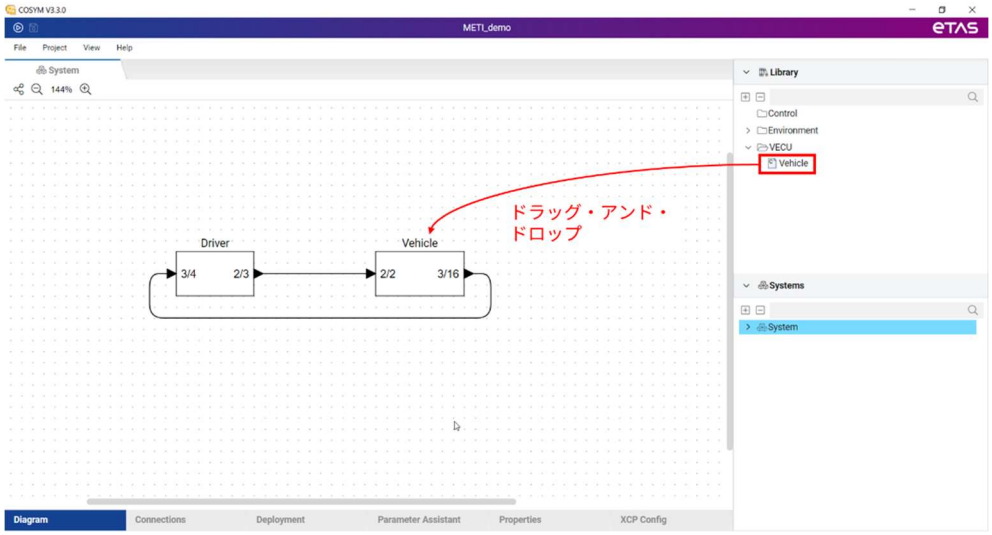


図 1.8.6 シミュレーションするシステムへの FMU の配置

そして、信号の接続やタスクの実行順序を設定します。

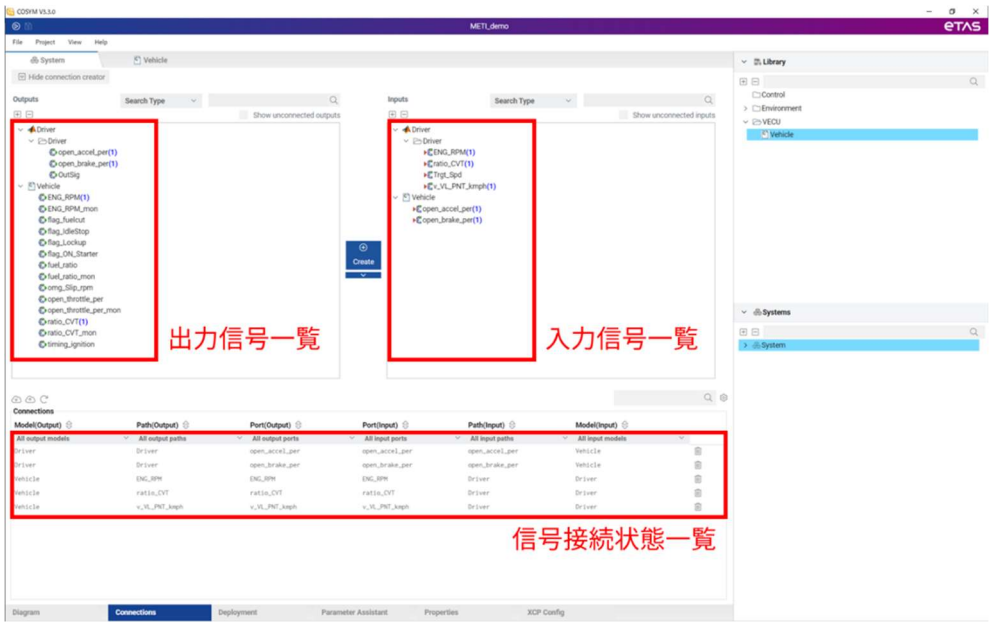


図 1.8.7 ブロック間の信号の接続

以上が完了したら、「Build and Run」をクリックします。シミュレーションの実験画面が起動し、前画面で設計したシミュレーション・システムの動作を確認することができます。なお、入力信号の操作やモニタ対象の信号の表示などの GUI の構成もこの実験画面で行います。

図 1.8.8 シミュレーション実行画面

1.9. モデロン　Modelon Impact

Modelon Impact は、モデロン社が開発した Modelica 言語をサポートするシステムシミュレーションツールです。Modelon Impact ではすべてのコンパイル済みのモデルは FMI の標準に準拠しており、FMI に対応したモデルの作成、統合、シミュレーションを行うことができます。主に以下の機能を有しています。

- FMI 2.0 をサポートします。
- Co-Simulation と Model Exchange（エクスポートのみ） をサポートします。
- Windows および Linux（特定コマンド）を対応しています。

また、Modelon Impact のカーネルである OPTIMICA Compiler Toolkit と利用した場合、FMI への対応する機能をより発揮できます。

- FMI 1.0 及び 2.0 をサポートします。
- Co-Simulation と Model Exchange をサポートします。
- Windows および Linux を対応しています。

1.9.1. FMU の作成

Modelon Impact のワークスペース（Workspace）にあるモデルから FMU を生成する場合、対象モデルを右クリックし、Export FMU を選択します（図 1.9.1）。

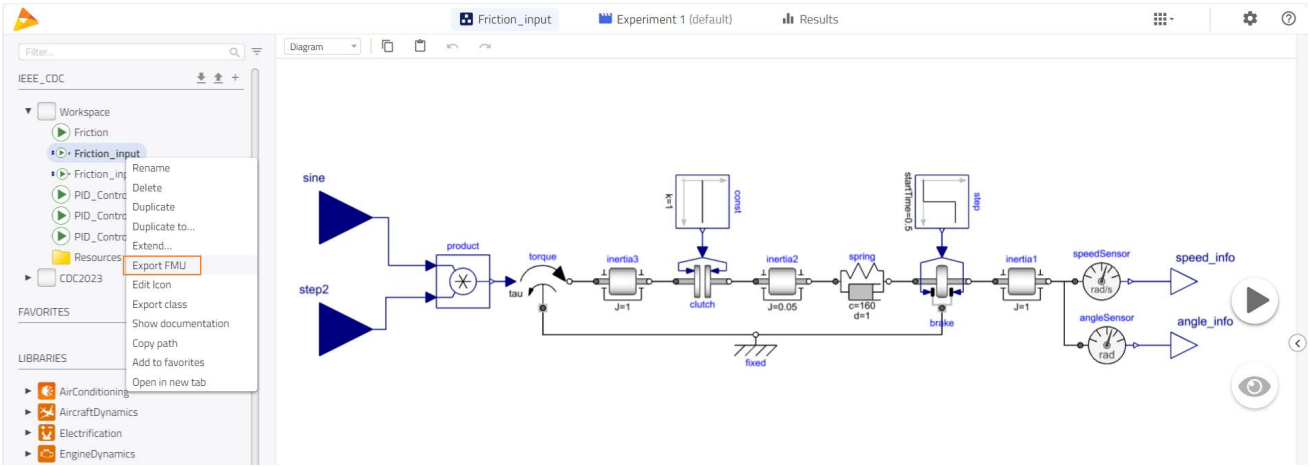


図 1.9.1　FMU 生成

次の画面（図 1.9.2）では、FMU 生成に関するオプションが表示されます。その内、FMU の種類（Model Exchange のみ、 Co-simulation のみ）を選びます。Co-simulation の FMU を選んだ場合、ソルバ設定がコマンドを実行した時のソルバ状態になります。

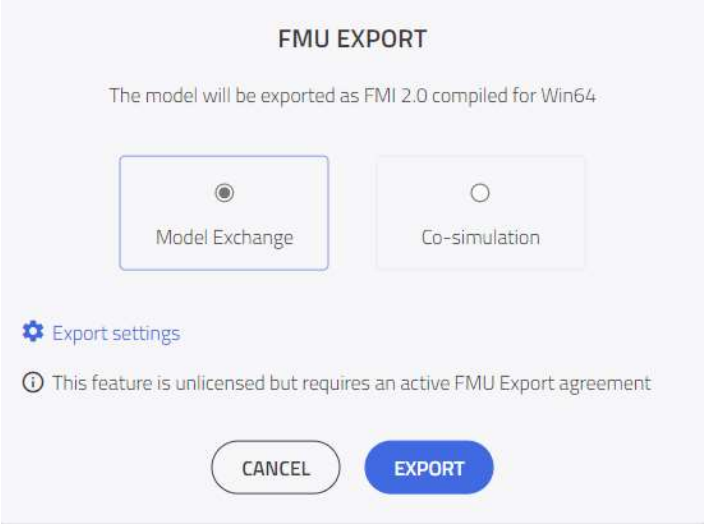


図 1.9.2　FMU エクスポート設定

図 1.9.3 の Export settings により、ほかのオプション設定ができます。

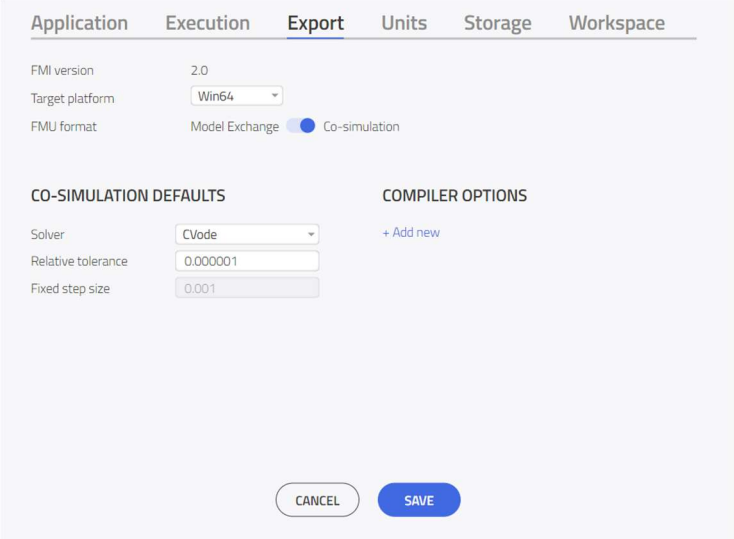


図 1.9.3　FMU エクスポートの詳細設定

1.9.2.　FMU のインポートと実行

FMU をインポートする場合、左上のワークスペースブラウザにあるインポートボタンにより FMU モデル（Co-simulation 2.0 のみ）をインポートします（図 1.9.4）。

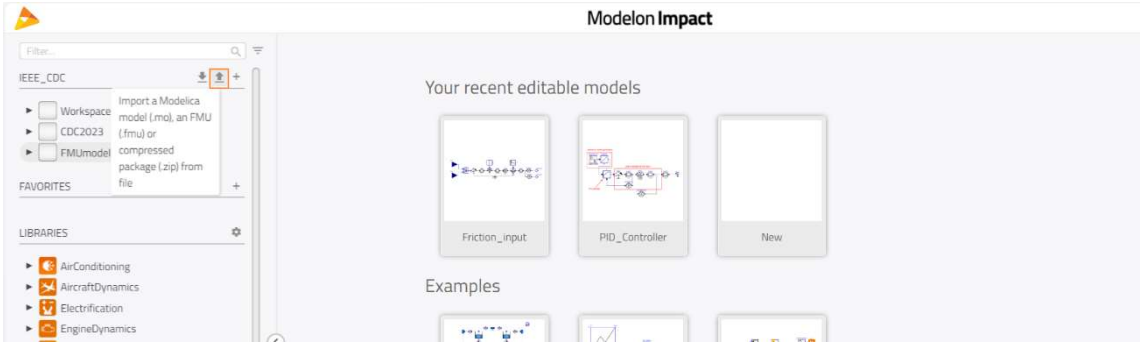


図 1.9.4　FMU インポート

インポート先のパッケージ名を選択して、モデルをインポートします（図 1.9.5）。

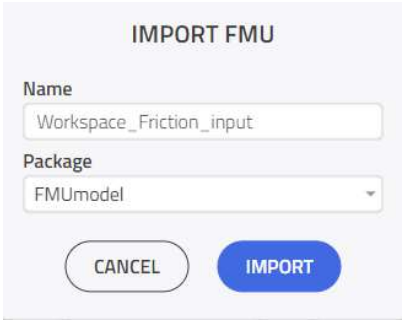


図 1.9.5　FMU インポート設定

FMU を実行するには、インポートされたモデルをモデルキャンバスにドラッグアンドドロップして追加します（図 1.9.6）。

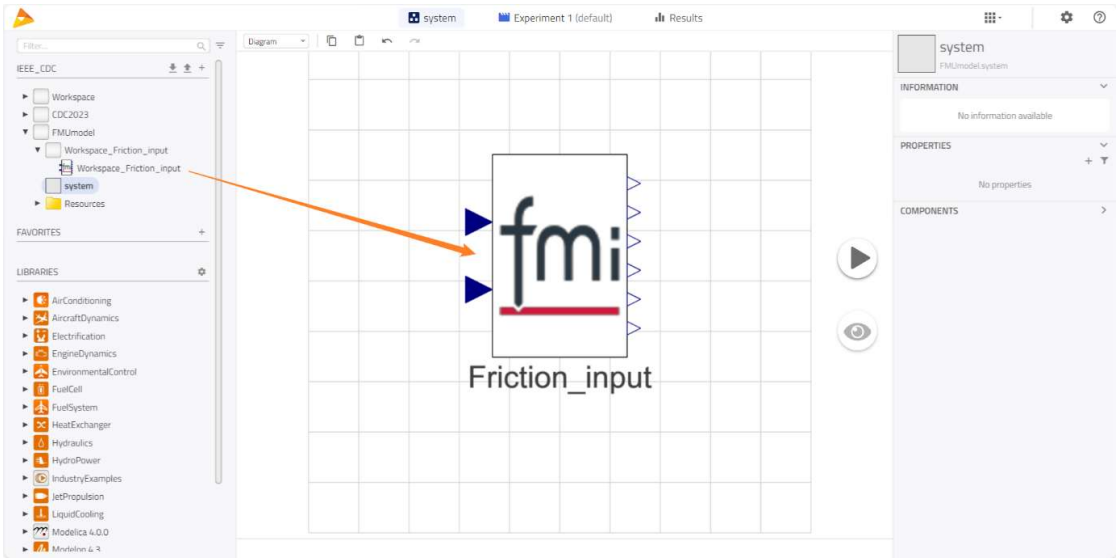


図 1.9.6　FMU を使用するモデル例

コンポネントにダブルクリックすると、パラメータを設定する画面が現れ、INFORMATION タブではモデルの情報が確認できます。PROPERTIES タブでパラメータを設定します。パラメータがタブごとに分類され、モデルのパラメータと FMU モデルのパラメータが表示されます（図 1.9.7）。

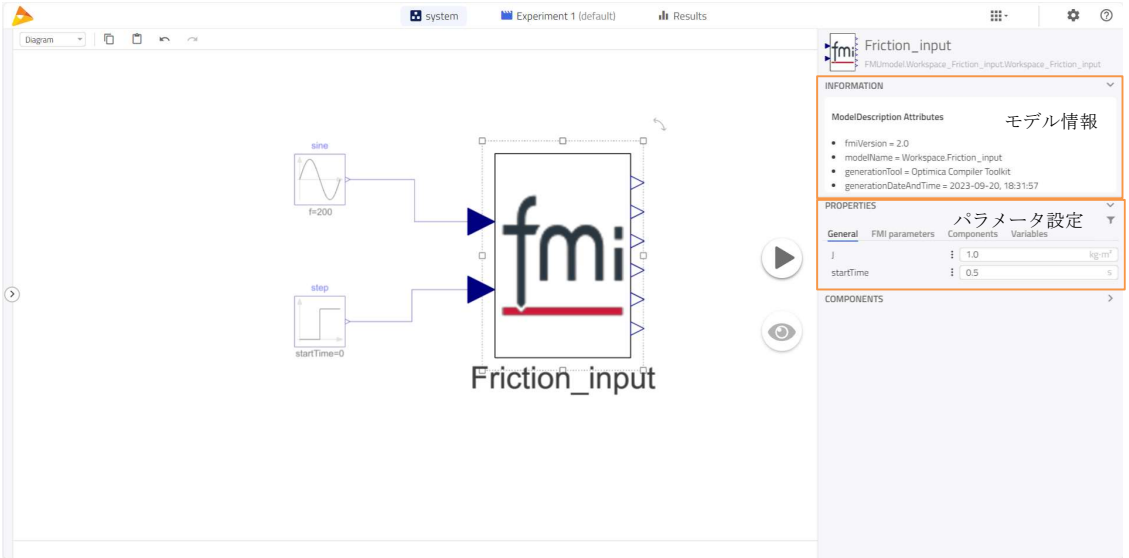


図 1.9.7　FMU のパラメータ設定

インポートされた FMU を、他の FMU および Modelica のコンポネントと結合して、Modelon Impact でのシミュレーションを実行して結果表示と解析ができます。

また、複数の FMU モジュールを結合して、システムモデルを作成することができます。図 1.9.8 に示したのはその一例です。

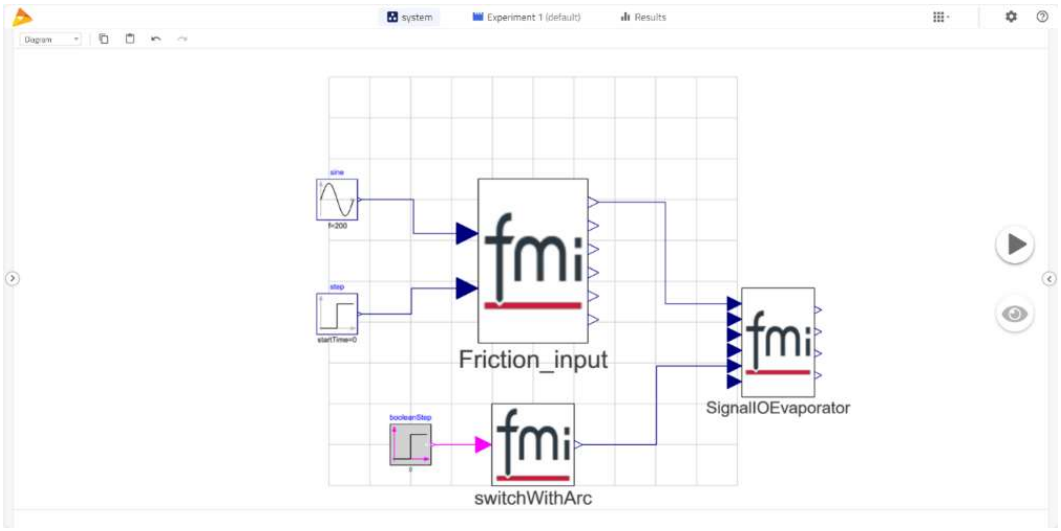


図 1.9.8　複数 FMU で構成したシステムモデル例

1.9.3.　OPTIMICA Compiler Toolkit の FMI サポート機能

OPTIMICA Compiler Toolkit（略して OCT）は Modelon Impact で使用される計算エンジンです。Modelica コンパイラとソルバを備えており、最適化や定常状態の計算など、ダイナミックシミュレーション以外の機能を提供します。Modelon Impact では OCT をカーネルとしての Jupyter Notebook と統合し、FMU モデル解析と結果分析を行います。

- FMI 1.0 及び FMI 2.0 をサポートします。
- Co-Simulation と Model Exchange をサポートします。
- Windows および Linux を対応しています。

Modelon Impact の UI から *JupyterLab* にアクセスし、ノートブックを作成します（図 1.9.9）。

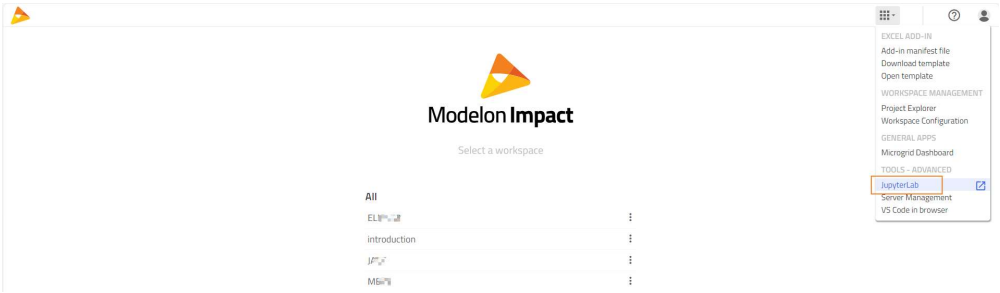


図 1.9.9　JupyterLab を呼び出す

ノートブックは Modelon Impact と繋がって、ワークスペースにあるモデルを読み取ることができます。Python コードよりモデルの解析を実行します。

（a）Modelon Impact のワークスペースと連携し、解析対象となるモデルを読取ります。

```
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）FMU モデルを生成します。ここでは FMU モデルの種類を設定します。デフォルトは Model Exchange の 2.0 タイプとなります。生成可能な FMU モデルは 1.0，または 2.0 バージョン。そして Modelon Exchange，Co-simulation，または ME+CS タイプが選びます。

```
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） 生成された FMU モデルをエクスポートして、解析を行います。

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

（d） 二つのモデルを繋がります。（Co-Simulation ワークフロー）

```
# 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）インポート数値を算出して入力します。

```
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） 解析を実行し、結果を確認します（図 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()
```

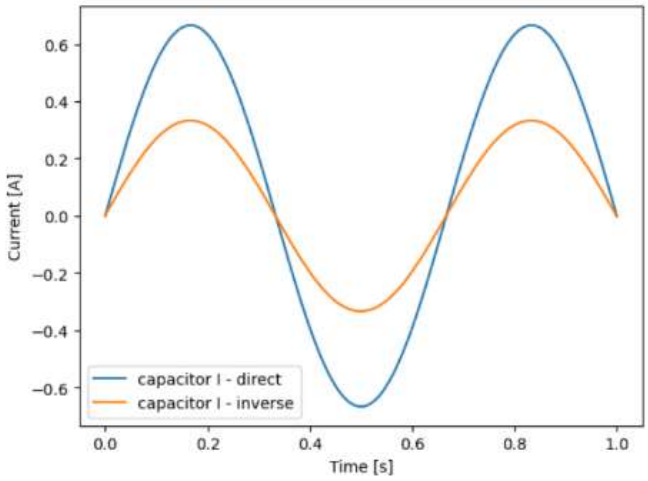
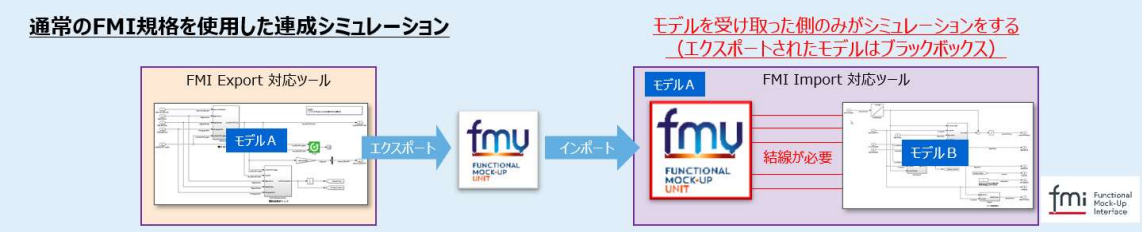


図 1.9.10　Jupyter Notebook での FMU 解析結果

1.10. VenetDCP

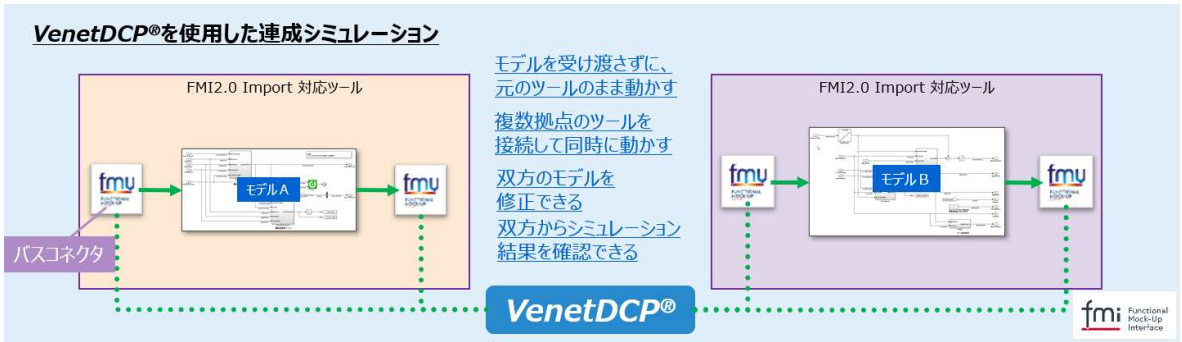
VenetDCP は、東芝デジタルソリューションズ株式会社が提供する分散・連成シミュレーションツールで、種類やバージョンの異なるシミュレーションツールを接続することができます。

通常の FMI 規格を使用した連成シミュレーションでは、モデルを FMU にエクスポートして相手に受け渡す必要があります。この場合、エクスポートされたモデルはブラックボックスとなり、モデルを受け取った側のみがシミュレーションできます。

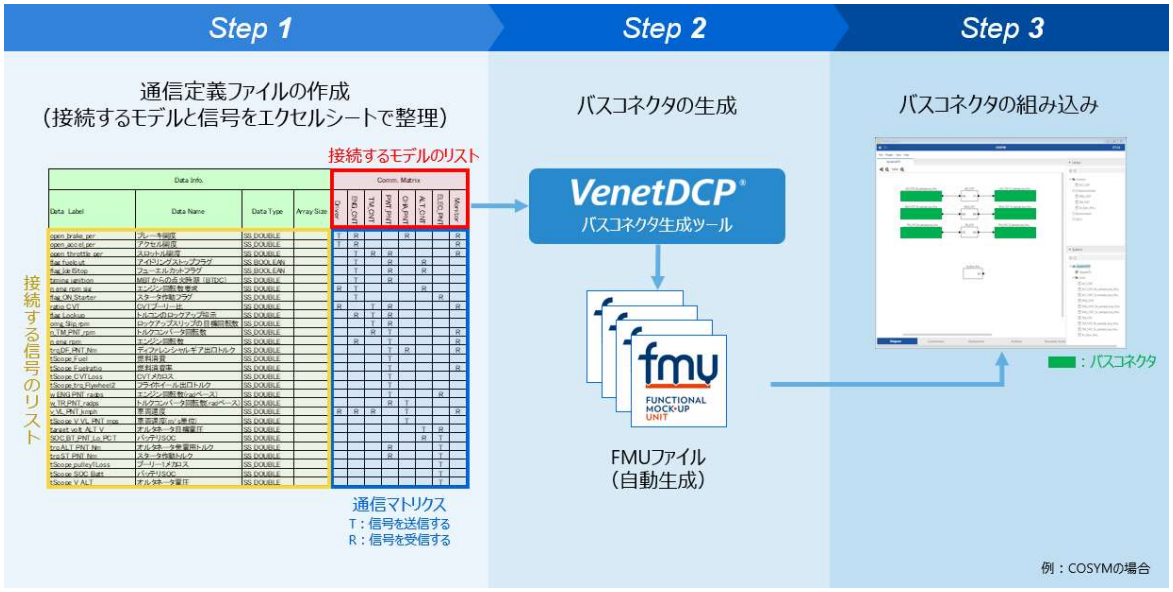


VenetDCP を使用した連成シミュレーションでは、モデルを受け渡さずに元のシミュレーションツールに置いたままシミュレーションできます。この場合、各モデルには VenetDCP が生成する FMI 通信機能ブロック「バスコネクタ」を組み込む必要があります。このバスコネクタは、FMI2.0 形式の FMU で作られており、FMI2.0 インポートに対応するシミュレーションツールで使用できます（FMI3.0 のサポートも計画しています）。

バスコネクタを組み込んだモデルでシミュレーションを開始すると、そのモデルはバスコネクタを経由して他のモデルとネットワーク越しに接続します。シミュレーション中、VenetDCP は接続された各モデルを、時刻同期させながら同時並行的に動作させます。これにより、複数のシミュレーションツールを複数拠点間で連携させた分散・連成シミュレーションが可能になります。また、モデルが FMU にエクスポートされていないため双方のモデルを修正でき、モデルが手元にあるため双方のシミュレーションツールで結果を確認できます。



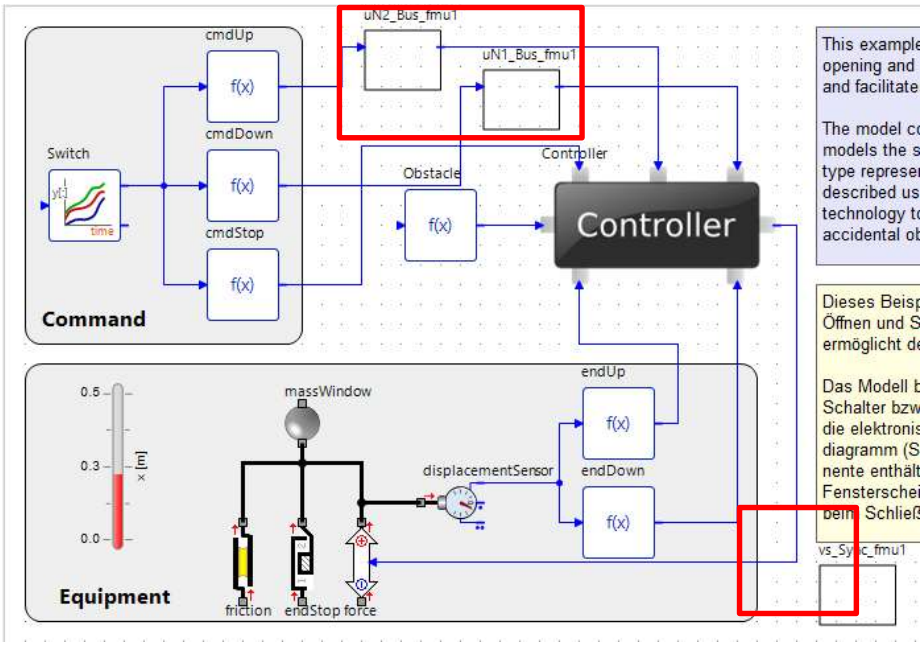
VenetDCP を用いた分散・連成シミュレーションの準備には、以下の 3 つのステップがあります。



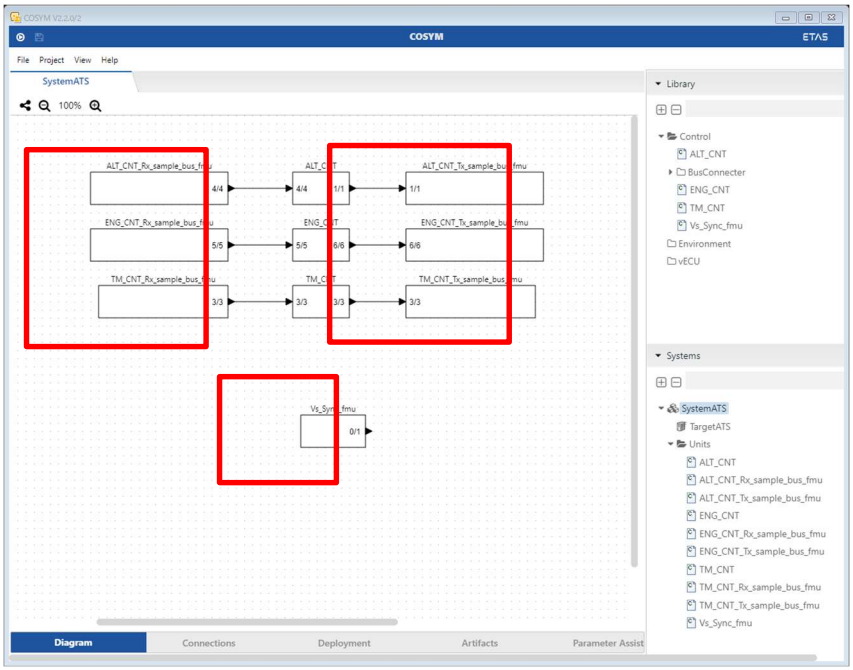
1. 「通信定義ファイル」を作成します。通信定義ファイルには、各モデルが持つ入力信号、出力信号、通信マトリクス（モデル間の信号結線）を記載します。
2. 「バスコネクタ生成」ツールに通信定義ファイルを入力し、バスコネクタを生成します。
   生成したバスコネクタを、分散・連成シミュレーションの参加者に配布します。
3. バスコネクタをモデルに組み込みます。各ツールの FMI インポート手順に従ってください。

以下は、バスコネクタを組み込んだモデルの例です。

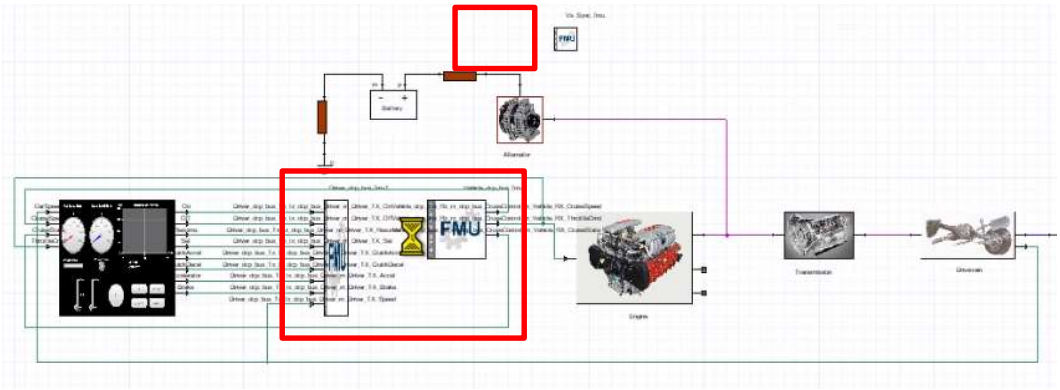
・接続例１．Simulation X との接続



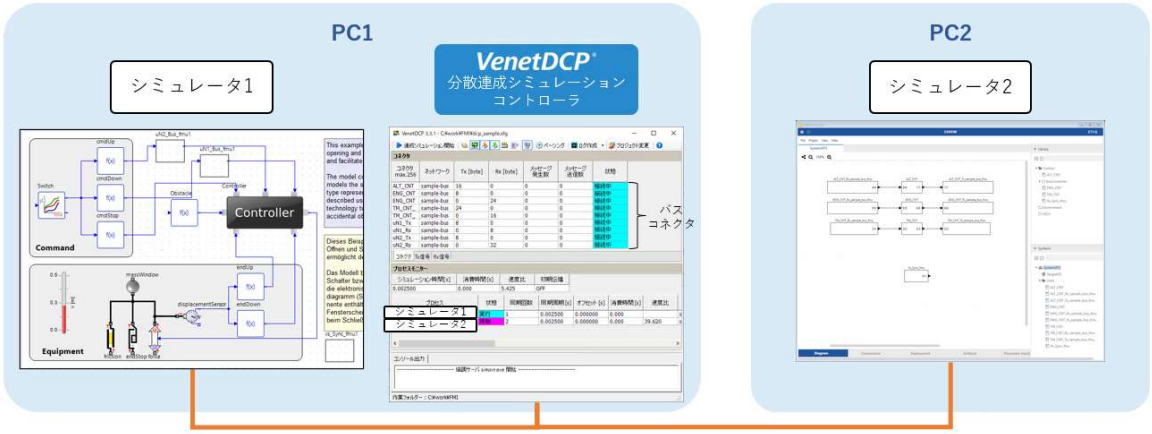
・接続例２．ETAS COSYM



・接続例３．アンシス Twin Builder



バスコネクタを組み込んだモデルでシミュレーションを開始すると、バスコネクタは自動的に「分散連成シミュレーションコントローラ」に接続されます。すべてのバスコネクタが接続されることで、分散・連成シミュレーションが開始されます。



1.11. OpenModelica

OpenModelica は、Open Source Modelica Consortium で開発されている Modelica 言語を使用したシミュレーションツールです。このツールは、無償で公開されています。今回は、バージョン 1.21.0 での紹介です。本バージョンが有する機能について、表 1.11.1 に示します。

表 1.11.1　OpenModelica FMI インタフェース機能一覧

FMI 作成機能	
FMI バージョン	1.0、 2.0
形式	Model Exchange、 Co-Simulation
OS	Windows、Linux、macOS
ライセンス	FMU 作成時：不要
	作成した FMU の実行時：不要
FMI 読込機能	
FMI バージョン	2.0
形式	Model Exchange
OS	Windows、Linux、macOS
ライセンス	FMU 読込時：不要
	読み込んだ FMU の実行時：作成側に依存

1.11.1. FMU の作成

FMU の作成方法についてご紹介します。事前にオプションで保存場所等の設定が必要になります。

保存場所は、ツール>オプション>全般にて、作業ディレクトリを指定します。

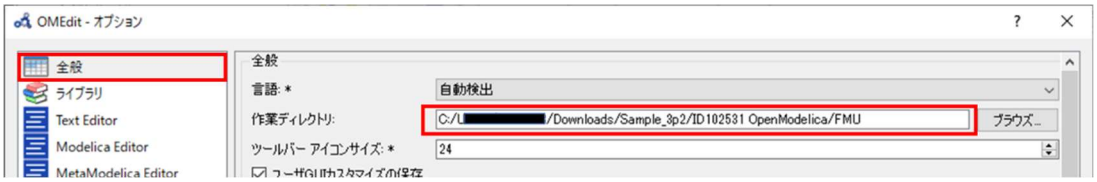


図 1.11.1　OMEdit-オプションにて作業ディレクトリを設定

FMI 1.0 と 2.0、もしくは Co-Simulation と Model Exchange の指定は、ツール>オプション>FMI にて、書き出しのバージョンとタイプを指定します。

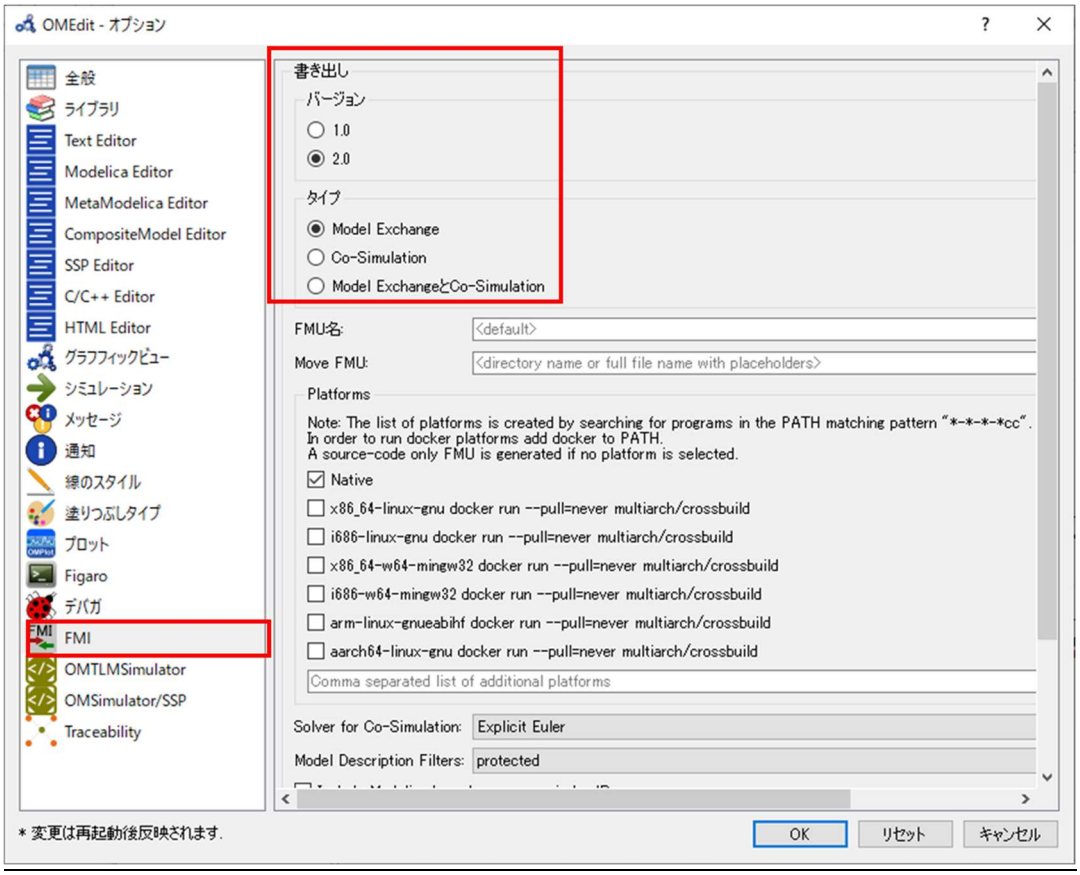


図 1.11.2　OMEdit-オプションにて FMI を設定

ファイル>書き出し>FMU を選択すると、FMU が作成されます。

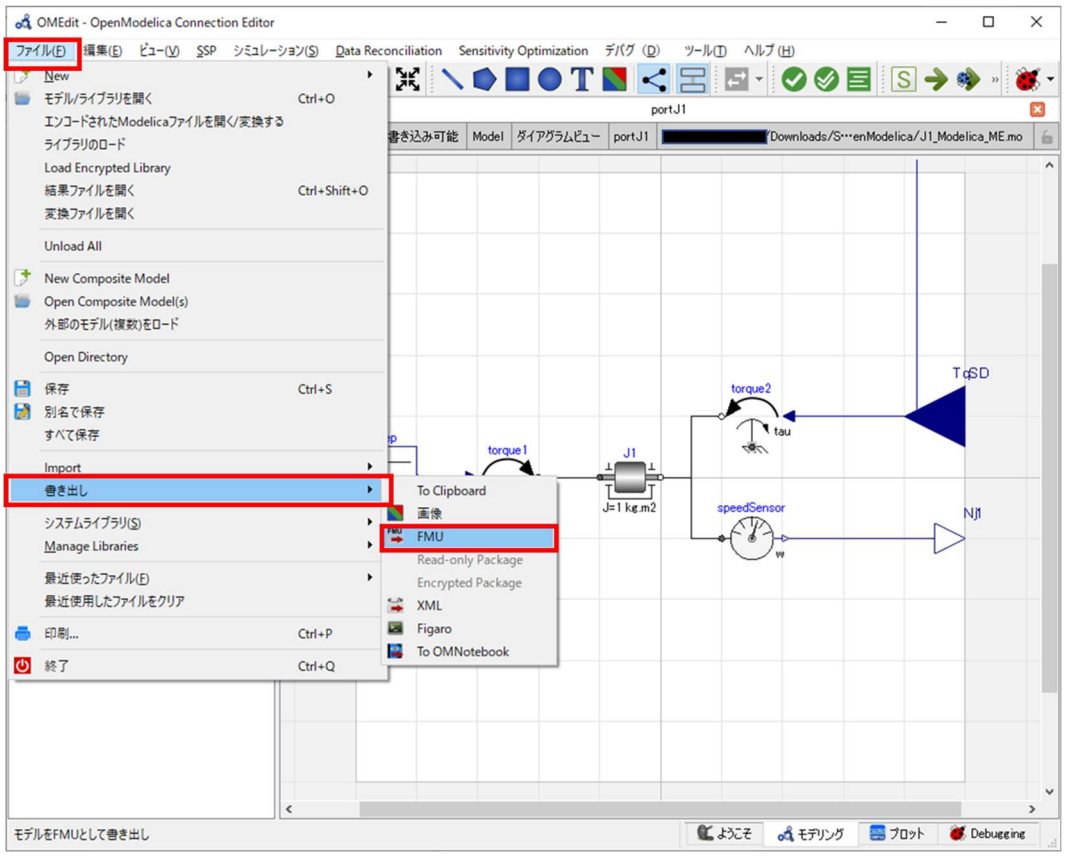


図 1.11.3　FMU へアクセス

メッセージブラウザにて、書き出し作業の結果（成功・失敗）が確認できます。成功の場合は、「The FMU is generated at ～」と表示されます。

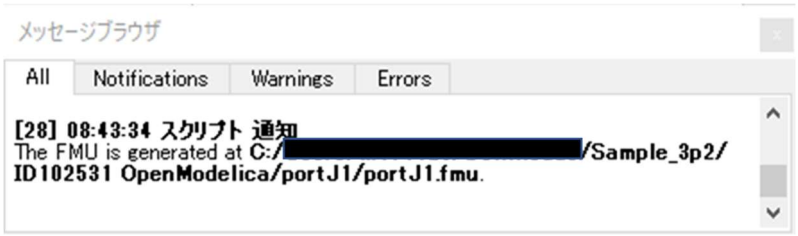


図 1.11.4　メッセージブラウザでの作業結果の確認

1.11.2. FMU のインポート

FMU のインポートについてご紹介します。ただし、インポートした FMU が実行できないケースが多々ありますので、今回は実行まででできたケースを元にご説明します。インポートをする前に、ツール>オプション>シミュレーションにて、「Enable FMU Import」にチェックを入れて有効にします。

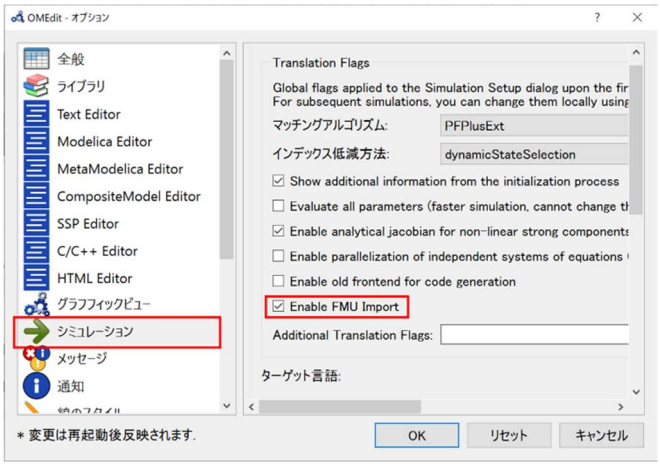


図 1.11.5　Enable FMU Import を有効化

ファイル>Import>FMU を選択します。

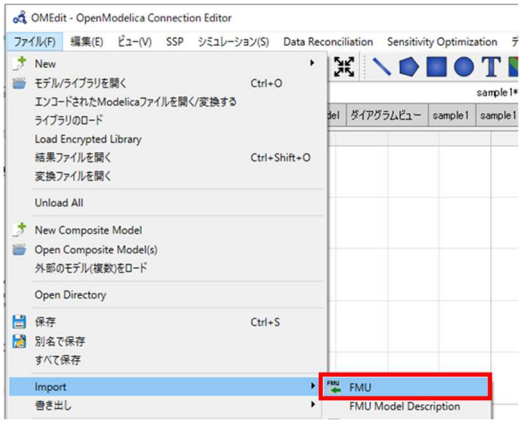


図 1.11.6　インポート機能へアクセス

次に、取り込みを行う.fmu ファイルを指定します。

図 1.11.7　取り込みを行う.fmu ファイルの指定

インポートした FMU は、ライブラリに表示されます。手動でダイアグラムビューに配置します。

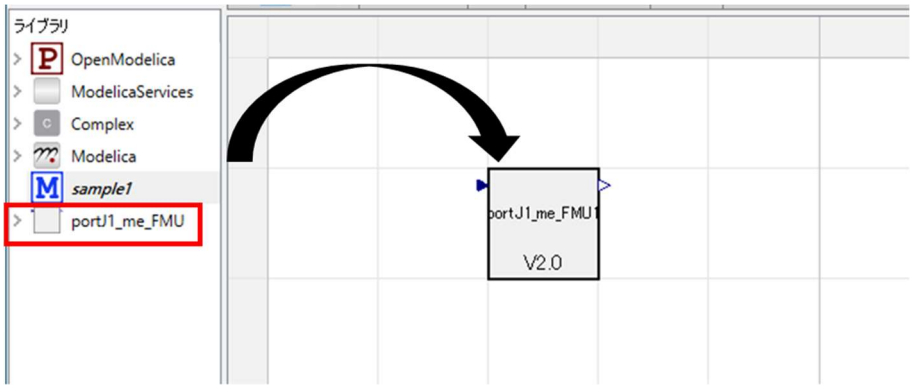


図 1.11.8　インポートした FMU

1.11.3. FMU の実行

モデリング>シミュレートにてシミュレーションを実行します。

図 1.11.9　シミュレーションの実行

プロット>変数ブラウザにて観測したい変数にチェックを入れます。これにより選択した変数の結果が表示されます。

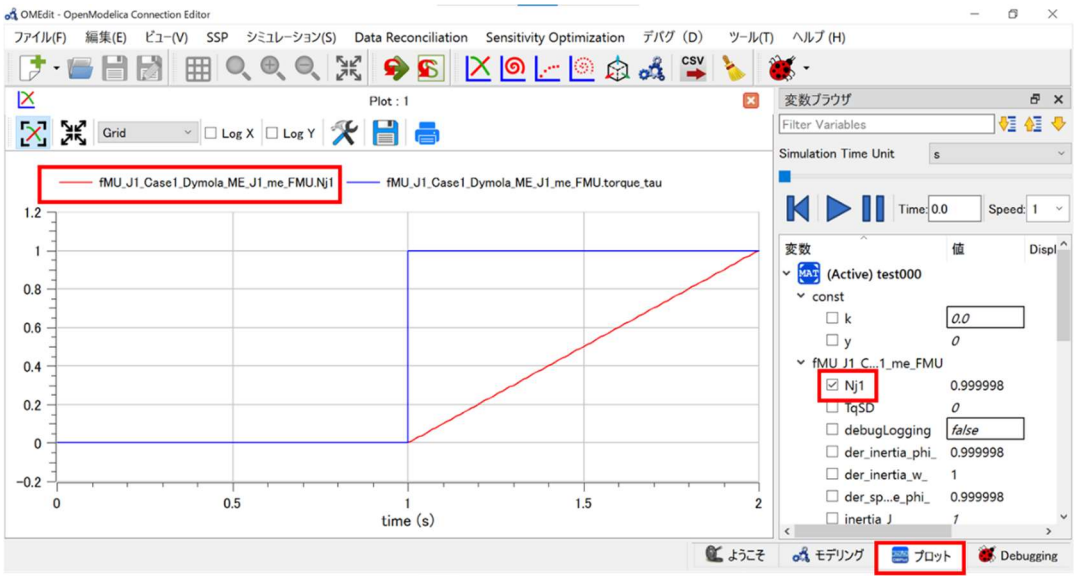


図 1.11.10　出力結果の選択と表示

1.12. FMPy

FMPy は、次の機能を持つ無料の Python ライブラリであり、Functional Mock-up Units (FMU) のシミュレーションを行うことができます：

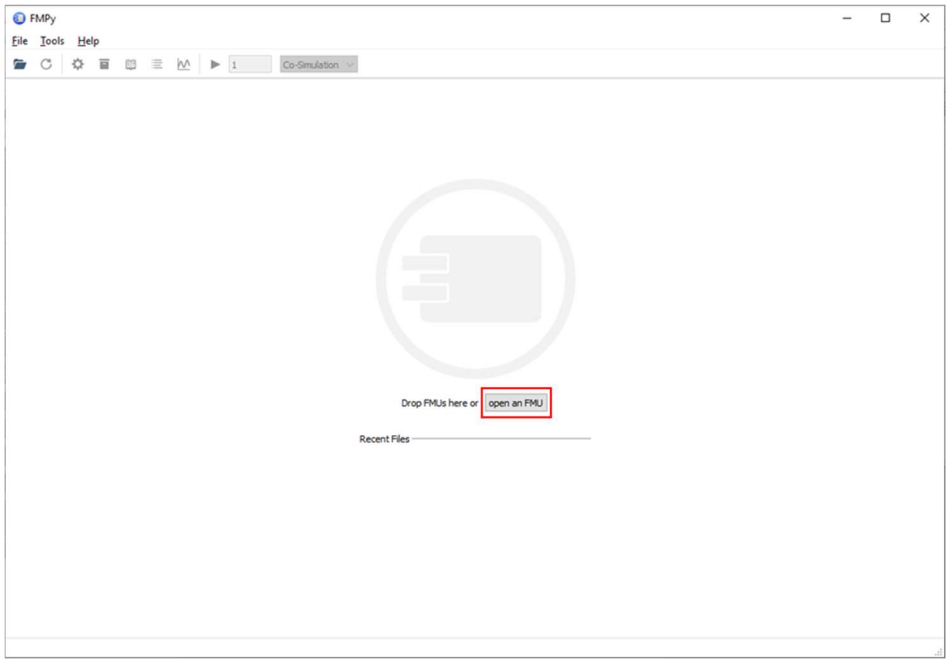
- FMI 1.0、2.0、および 3.0（ベータ） をサポートします。
- Co-Simulation と Model Exchange をサポートします。
- Windows、Linux、および macOS で実行できます。
- コマンドライン、グラフィカルユーザーインターフェース、および Web アプリケーションを提供します。
- Jupyter Notebooks を作成できます。
- C 言語のコード FMU をコンパイルし、デバッグ用の CMake プロジェクトを生成することができます。

FMPy は基本的に単体の FMU のインポートしか対応していません。複数の FMU をつなげる場合は、Dymola のようなソリューションをご利用ください。

GUI を起動しましょう。

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

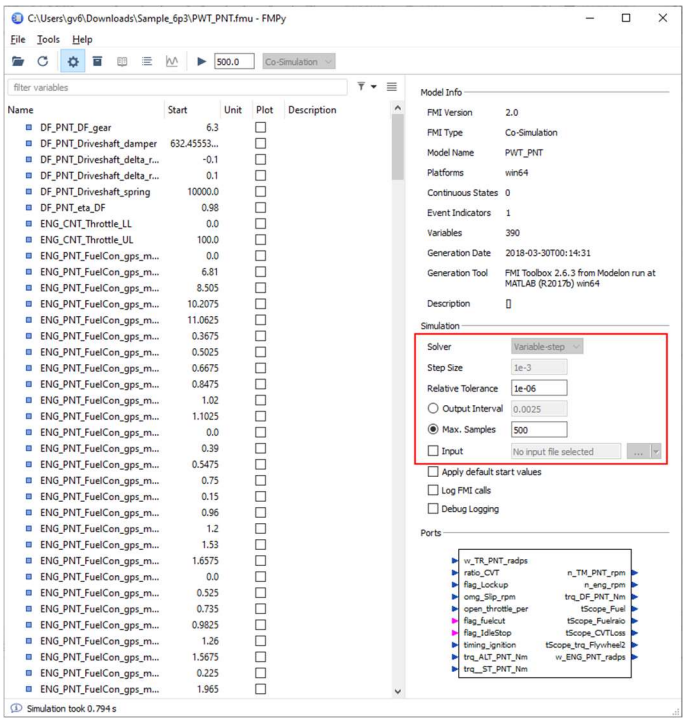
下記の画面が表示されます。



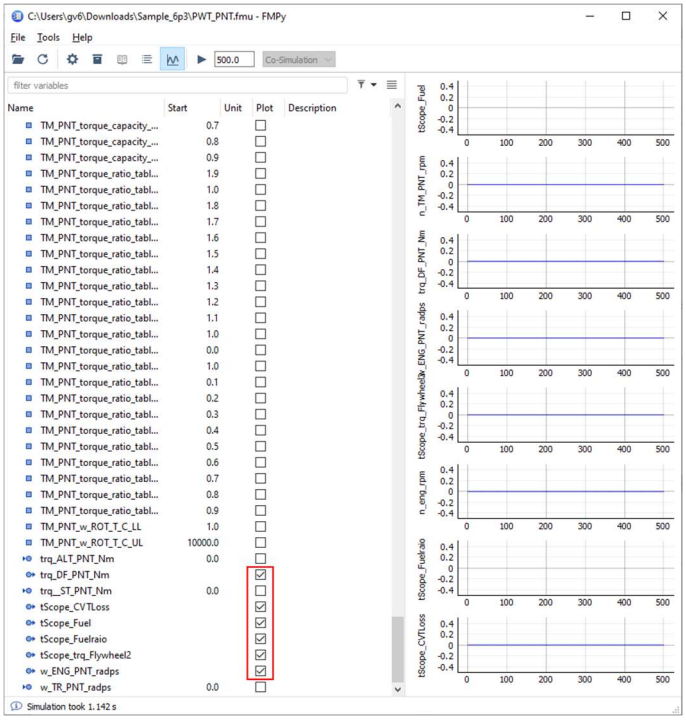
“open a FMU” ボタンまたはドラッグアンドドロップで FMU をロードしてください。

シミュレーション期間またはソルバのパラメータを UI 上で指定できます。

もしインプットファイルを利用したければ、右側に”Input”と言う項目をチェックして、CSV ファイルを指定することもできます。



表示したい変数を選択し再生ボタンをクリックすると、プロット画面が現われます。



第 2 章　FMI 活用事例

この章では、FMI の活用事例を紹介します

2.1. ハイブリッド航空機に関する FMI 活用モデル

2021 年の「知の拠点あいち 重点研究プロジェクト」の一環として，名古屋大学と三菱重工航空エンジン株式会社により作成した，ハイブリッド航空機モデルを例題にチュートリアルを進めていきます.

2.1.1. サンプルモデルの説明

本航空機モデルは，マスタツール(Simplorer) 内で作成したバッテリモデルと，電動駆動モデル(Modelica)，油圧系統・ダイナミクスモデル（Amesim）の計 2 つの FMU で構成して Model Exchange を行います．概略は図 2.1.1 を確認してください.

OS は Windows 64bit 版，FMI2.0 のみに対応しています.

ただ本モデルは，権利の都合上サンプルモデルは公開できませんので，ご了承ください.

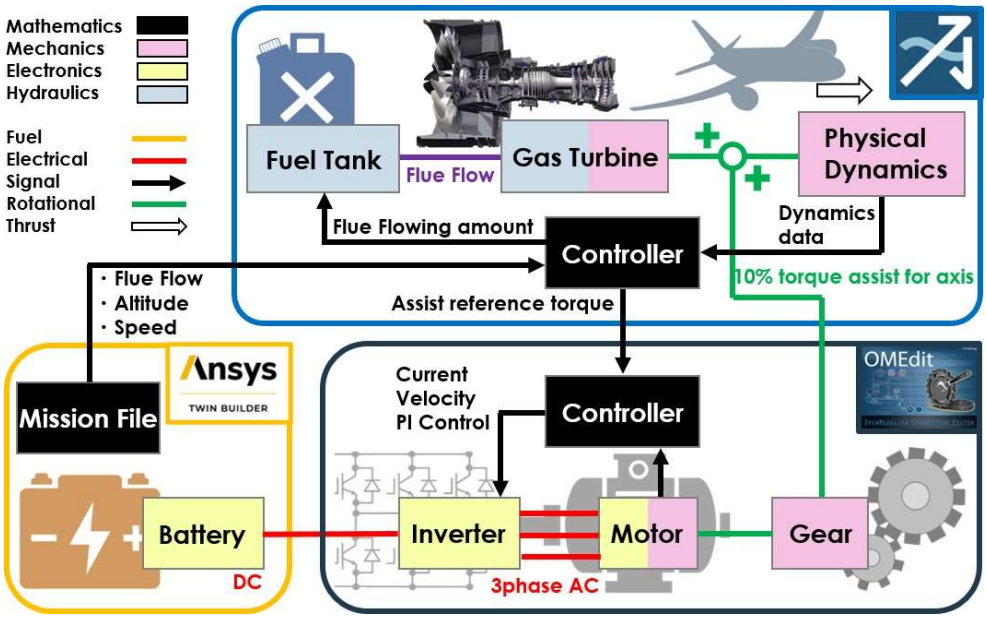


図 2.1.1 Hybrid aircraft's system simulation model

今回マスタツールには Simplorer を用いました．そこに 2 つの FMU を取り込みます．取り込み方法は，メニュー画面で指定する場合や Explorer からドラッグ＆ドロップなどツールによって異なりますので，マニュアルを参照してください.

また，本モデルはマスタツールにより，バッテリモデル・航空機の飛行データ（図 2.1.1 中 Mission File）を指定する必要がございますので，ご理解ください.

2.1.2. モデルの概要

2.1.2.1. 電気系統モデリング

電気系統モデリングではバッテリから受け取った直流電圧をインバータで三相交流に変換し，モータによりトルクとして出力する過程を数式モデルにより表現しました．図 2.1.2 に電気系統モデルのブロック図を示します．ここではバッテリ，インバータ，モータ，コントローラ，パワー半導体の損失に関して個々の内容を示します．

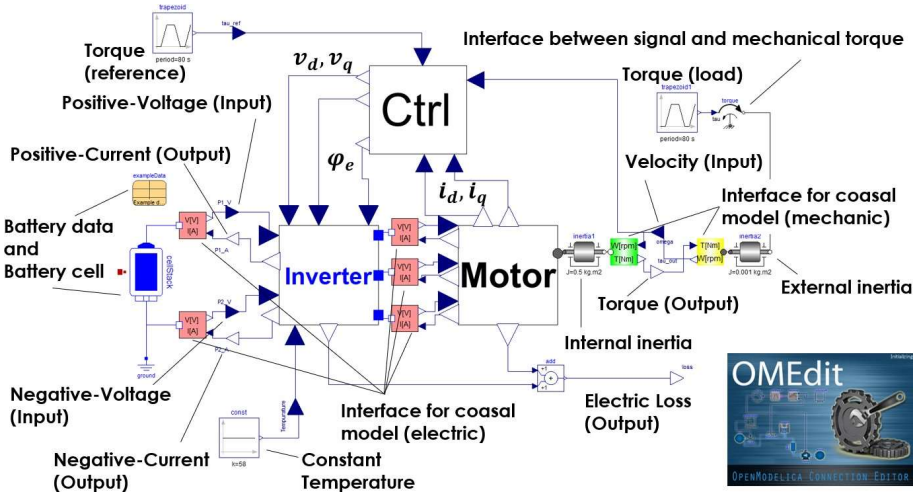


図 2.1.2 Electrical Drive Block Model

＜１：バッテリ＞

バッテリは，実用化されている二次電池の中で最もエネルギ密度が大きいリチウムイオン電池を電気的等価回路として表現したモデルを用いました．

＜２：インバータ＞

インバータは本来スイッチングを用いて DC/AC 変換を行います．ですが，本研究では計算コスト低減のため平均化法を用いて式 (1)のように表現しました．ただし，V_{DC}, I_{DC}はそれぞれ，インバータ入力の直流電圧・電流を示し，v_{ac}, i_{ac}はそれぞれ，出力の三相交流電圧・電流を示します．また，P_{inv}はインバータで発生する損失を示します．図 2.1.3 に平均化法とスイッチングを考慮した際の電圧波形を示します．平均化法とは，実際のスイッチングによる影響などを考慮せずに，理想的な出力だと仮定し，インバータ損失（P_{inv}）を別途計算することです．今回検討したスイッチング損失，導通損失は ＜５：パワー半導体＞ で詳細を述べます．

$$V_{DC} I_{DC} = i_{ac} v_{ac} + P_{inv} \tag{1}$$

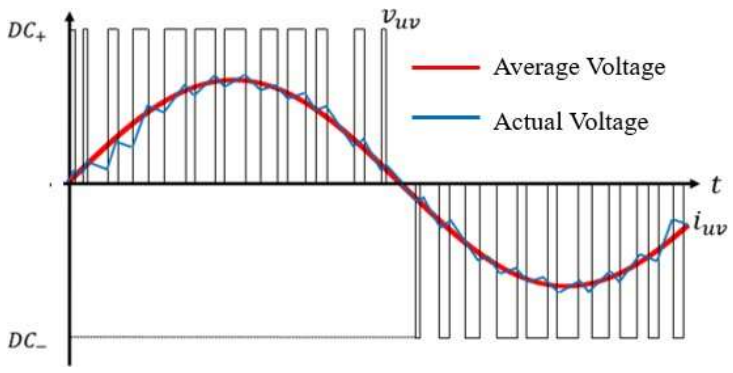


図 2.1.3 Model averaging

＜３：モータ＞

モータはインバータから受け取った三相交流電源をトルクへと変換する AC モータのうち埋込磁石同期モータ(IPMSM) を採用しました．本モータの特徴としては大トルク，高速回転を実現することができ，電気自動車でも採用されるモータです．ここでは式 (2) に dq 軸上での電圧方程式，式 (3) に IPMSM によって得られるトルク(τ_e) を示します．ただし，v_d，v_q を dq 軸電圧，R_a を電機子抵抗，i_d，i_q を dq 軸電流，L_d，L_q を dq 軸インダクタンス，ω_e を電気角速度，K_E をモータの鎖交磁束，p を極数とします．

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

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

＜４：コントローラ＞

コントローラでは指令トルクを受け取り，そこから回転速度，モータに流れる電流，モータに印加する電圧，パラメータ設定から最適な制御を選択し，非干渉電流制御を用いてインバータへと指令電圧を出力します．コントローラへの指令値（入力）は，航空機が指定されたフライトを達成するために必要な出力トルクの 10 分の 1 としました．これによりモータが，ガスタービンの出力をアシストするというパラレルハイブリッド方式として解析を行いました．また本研究では， IPMSM を採用したため，制御方式として図 2.1.4 のフローチャートを提案します．

図 2.1.5 に各制御における具体的な動作点を示します．ただし，横軸は d 軸電流（i_d）を，縦軸は q 軸電流（i_q）をそれぞれ示します．図 2.1.5(a) に示す最大トルク・電流制御は，低・中速時に，指令トルクを実現する最小の電流$|i_{dq}| = \sqrt{i_d^2 + i_q^2}$ を選択するアルゴリズムを構築しました．

図 2.1.5(b) に示す指令トルク-電圧制限制御では，(a) 最大トルク・電流制御点が制御範囲外となった際に，指令トルクを実現する最小電流 I を，指令トルク曲線と電圧制限曲線の交点を選択するアルゴリズムを構築しました．

図 2.1.5(c) に示す電流制限・電圧制限制御は，指令トルクが電流の制御可能範囲外となった際に最大のトルクを実現する電流動作点として，電流制限曲線と電圧制限曲線の交点を選択するアルゴリズムを構築しました．

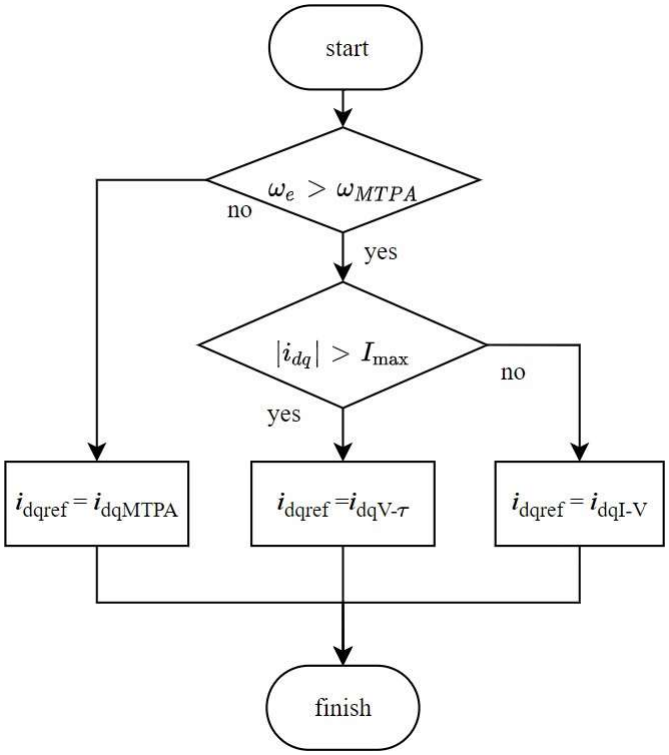


図 2.1.4 IPMSM control flow chert

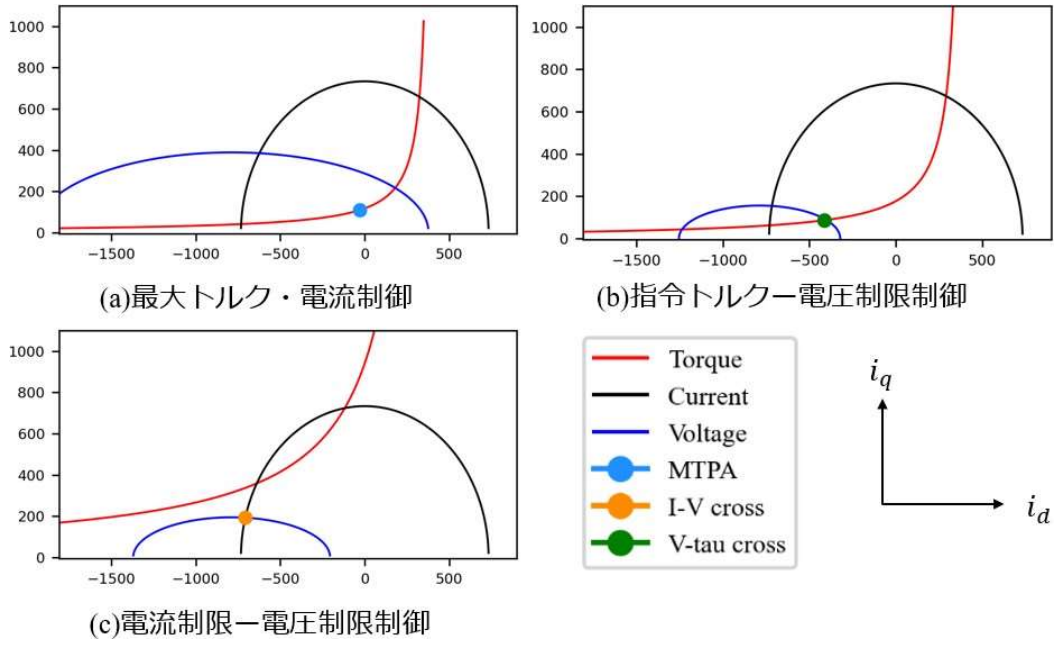


図 2.1.5 IPMSM control patterns

＜５：パワー半導体損失＞

パワー半導体の損失（P_{sem}）は，式 4 に示すように導通損失（P_{con}）とスイッチング損失（P_{sw}）の和として定義しました．まず導通損失は，式 5 に示す通り，パワー半導体に流れる電流（I_{ds}）とオン抵抗(R_{on}）より求められます．次にスイッチング損失に関しては，＜２：インバータ＞で述べた通り，DC/AC 変換とは別に計算をし，テーブルとして参照しました．

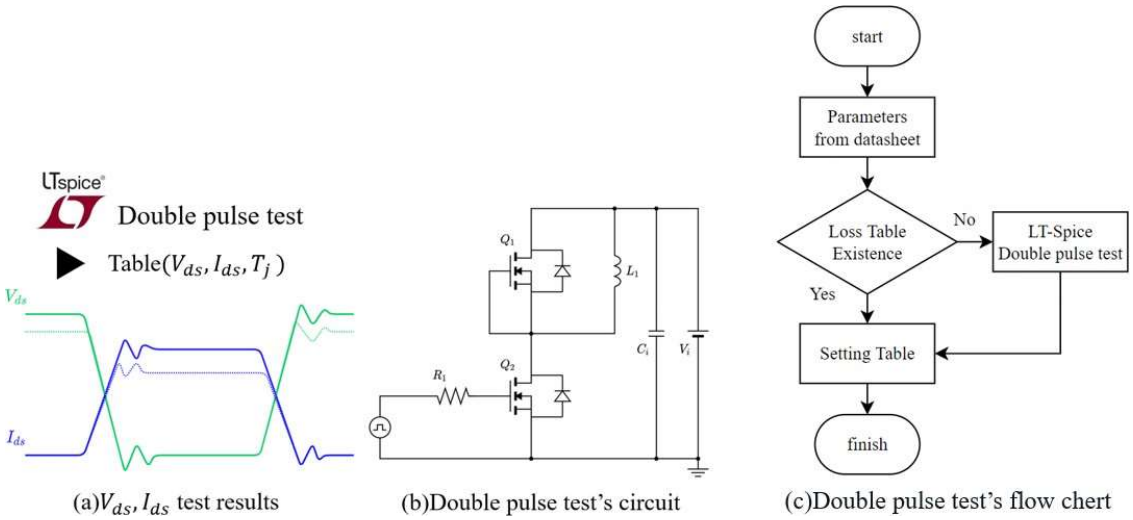
以下に詳細を述べます．まずデータシートからパワー半導体固有のパラメータを取得します．その後，当パワー半導体における損失テーブルが存在しているかどうかを判定し，損失テーブルがない場合，図 2.1.6(a) に示すダブルパルス試験回路を LT-Spice を用いて解析します．次に図 2.1.6(b) に解析結果を示します．この結果からオン側・オフ側に生じる損失を積分することにより求めました．以上から，式 6 に示すようにドレインソース電圧(V_{ds})・電流(I_{ds})，並びにパワー半導体のジャンクション温度(T_j) を引数に持ち，損失を出力するテーブル(Table(V_{ds}, I_{ds}, T_j)) を作成をいたしました．

図 2.1.6(c) にパワー半導体損失を計算するまでの手順を示します．この解析手法の利点としては，逐次スイッチングを考慮したモデル解析を必要とせず，解析時間短縮につながる点です．

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

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

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



(a)V_{ds}, I_{ds} test results　(b)Double pulse test's circuit　(c)Double pulse test's flow chert

図 2.1.6 Double pulse test

2.1.2.2. 航空機・ガスタービン系統モデリング

油圧系統のモデリングでは Amesim を用いてモデリングを行いました．概要を以下に示します．飛行ミッションプロファイル(速度，高度) を入力として，制御部で燃料流量，指令トルクを算出し，ハイブリッドエンジンへの入力としました．また，簡易推定した電気 系重量を加味し，航空機システムとして成立するよう，乗客人数や航続距離を調整しています．

2.1.2.3. モデルの接続

次に，図 2.1.7 にマスターツール(Simplorer)における，各 FMU モデルの接続関係を示します．こちらを参考に各 FMU と Simplorer 由来のバッテリ・飛行データを接続します．注意点として，すべての入出力は単位のないシグナル信号として扱われるため，注意が必要です．また，ガスタービンの回転速度は，マイナス値が駆動方向の回転となることにも注意が必要です．

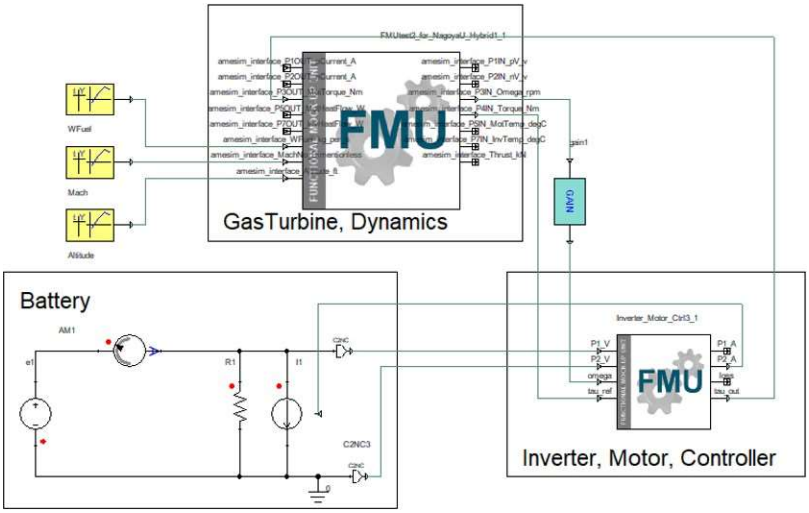


図 2.1.7 Model connection (@Master tool)

2.1.3. 航空機の飛行データ

飛行データは図 2.1.7 中左上の黄色ブロックで表現しました．これらは高度，燃料の流量，機体速度を，それぞれ指しており，図 2.1.8 に，乗客員数 100 名程度の中型機における典型的な飛行データを示します．このデータはコントローラへの指令値となります．

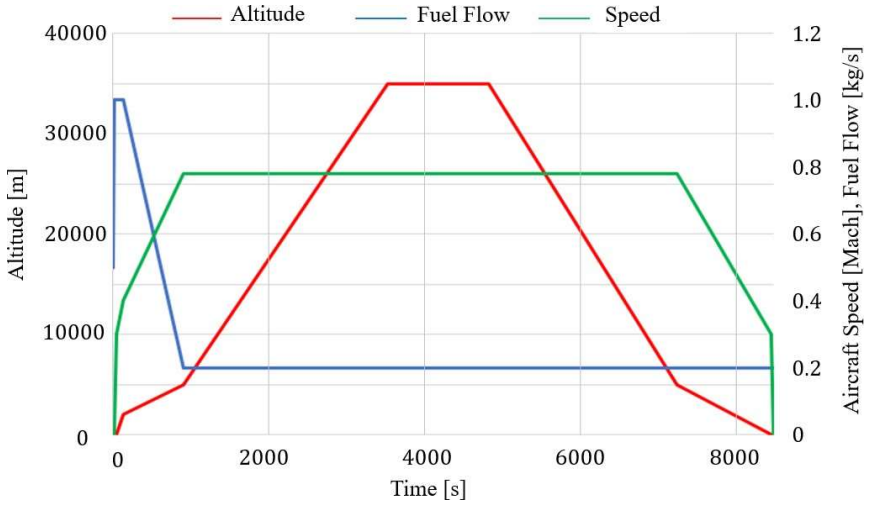


図 2.1.8 Flight profile (input data)

2.1.4. マスタツールのシミュレーション設定

図 2.1.9 に Simplorer におけるシミュレーションパラメータ設定画面を示します．こちらの画面で所要時間の 8430 秒をシミュレーション時間として設定します．

ステップサイズは，解析結果の時定数に合わせて可変にしております．下に示す最小・最大ステップサイズ値は初期値であり，図 2.1.9 に示します．この初期値は，時定数が大きく，または小さくなれば，それに合わせて変更するように設定しております．

Start Time　　0 *s*

Stop Time　　8430 *s*

Solver and Tolerances　　TwinBuilder,　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（出力結果のサンプル時間なので任意）

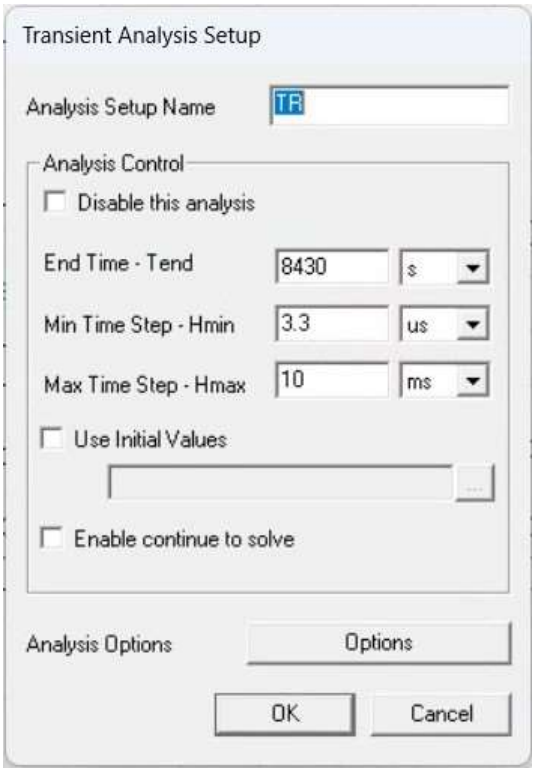


図 2.1.9 Simulation parameters setting シミュレーション結果

本モデルの解析結果を図 2.1.10 に示します．本解析では8430 s のフライトにおける解析を行いました．今回，航空機エンジンのトルクをモータにより10 % アシストするパラレルハイブリッド方式を取り燃料消費の激しい離陸開始時から1200 s までアシストした結果になります．ここで，ω_e をモータの回転速度[krad/s]，H を高度[kft]，T を機体にかかる推力[kN]，τ_e をモータの出力トルク［kNm]，v_M を速度（音速)[－（無次元)]とします．
図 2.1.10 より，解析開始時から1200 s までに機体にかかる推力が非常に高いことが確認できます．また，その区間においてガスタービンが生み出すトルクの10 % を指令値として，モータトルク（τ_e）がアシストしていることも同時に確認することができます．
本解析結果は，リージョナル機・中型機規模の運航において，飛行ミッションプロファイル(速度，高度)を与えた場合の結果であり，電気駆動機構の重量を加味しても，モータのアシストにより，航続距離を延ばすことができました．また，同規模のガスタービンのみの航空機におけるフライト結果と比較をしても，妥当な結果であると確認できました．

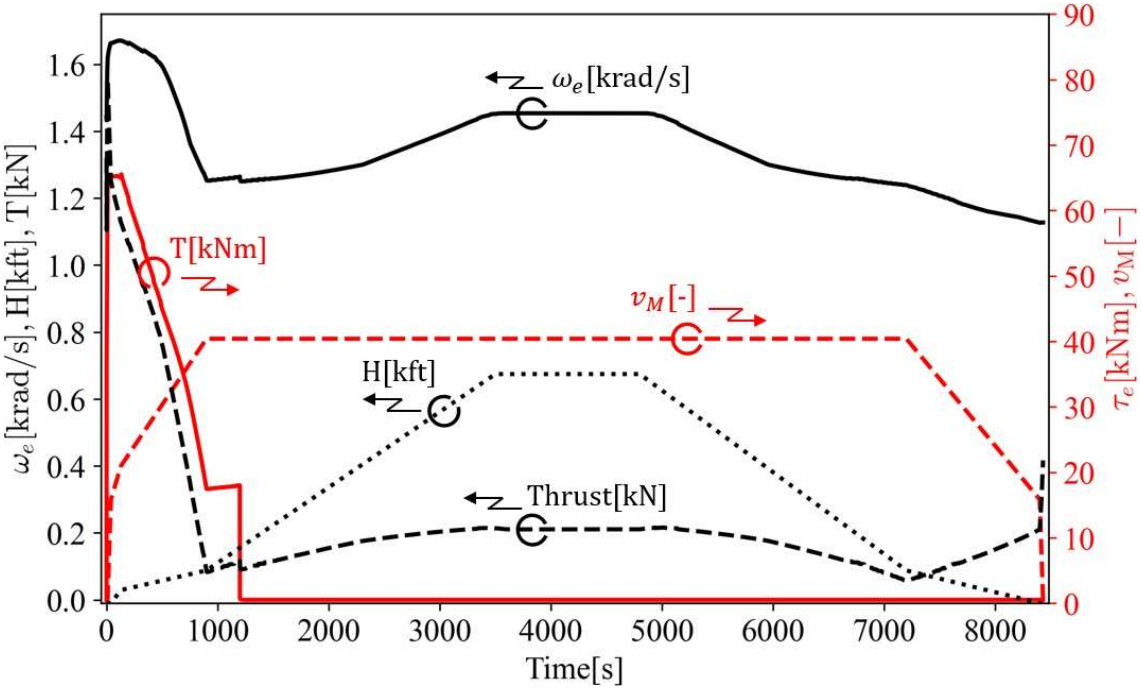


図 2.1.10 Simulation results

2.2. 1D-CAE と 3D-CAE の連成

2.2.1. 調査の目的 (2020 年～2023 年)

近年では 1D シミュレーションモデルを FMI 規約に基づいたモデル交換プロセスを活用して様々なツール間でのモデルの運用が成され、複雑な問題への適用が試みられています。今後、FMI を活用したモデルの流通を考慮すると、3D シミュレーションモデルと連携して機能抽出やコシミュレーションの需要が増すと予想されます。WG では幅広い活用のためにどのような情報の提供が求められているか調査し、その実現手法について検討し、その効果と課題を抽出することを目的としました。

2.2.2. 対象ツール種別の定義とツールの傾向

対象となるツールの種別として３種類に分類し(1)、FMI インタフェースを有する機能について調査しました。

A) 1D シミュレータ　基本方程式に対し常微分、代数微分方程式を用いて定義された要素モデルの構成を線形化し、線形代数方程式として求解する。機構解析もこれに相当する。

B) 3D シミュレータ　連続体力学に基づき有限要素法や差分法、ボクセル法などを用いて３次元構造を離散化して離散方程式を解く。機構解析と弾性体計算をカップリングしてそちらを主とする場合には 3D と分類する。

C) リアルタイムシミュレータ　リアルタイム環境での実行を念頭に、SILS、 HILS 上でプラントモデルのエミュレータや制御アルゴリズムの実装として運用されるものとする。

FMI ホームページ(2)では FMI インタフェースを有する各種ツールについて情報を公開しています。ツール種別と適用分野に関して抜粋したものを表 2.2.1 に示します。なお、本表ではシステムシミュレーションツールと機構解析だけを有するツール、機能確認が取れなかったものは除外しています。

マルチボディシミュレータでの対応が最も多く事例も豊富ですが、流体解析（CFD）、構造解析（Structure）分野での利用も進んでいる事が確認できます。表中 PIDO（Process Integration and Design Optimization）は最適化や統計分析を主題とするプロセス統合ツールを指します。

表 2.2.1 FMI 対応 CAE ツールの種別と数（2000 年）

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

これ等の調査を踏まえ、ツール連携の利用方法を調査したところ、概ね図 2.2.1 に示す４つのユースケースに分類されました。

① 3D CAE ツールを Co-Simulation（CS）のスレーブとして 1D システムシミュレーションと連携。通信などを用いて 3D ツールのソルバを利用。

② 1D システムを 3D ツールへの境界条件として Import して連携。

③ 最適化やプロセス統合ツールを用いて導出した 3D モデルの特性情報を縮約し、1D システムやリアルタイムシミュレータにて運用。

④　リアルタイム環境で制御、システムシミュレーションモデルを運用するための連携。

3D – 3D 間の連携は調査の範囲内で確認することが出来ていません。

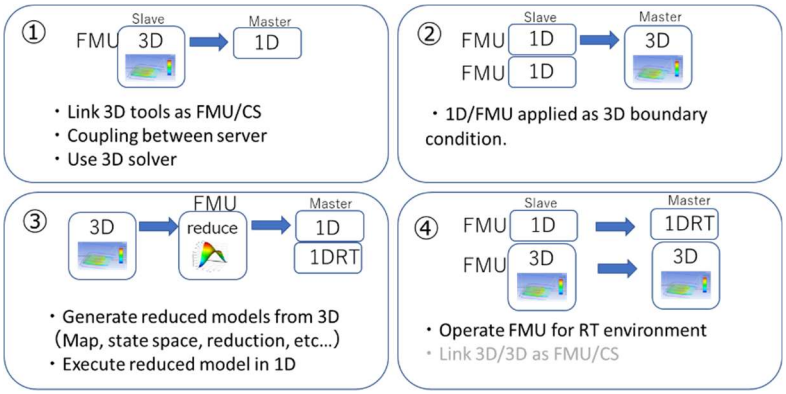
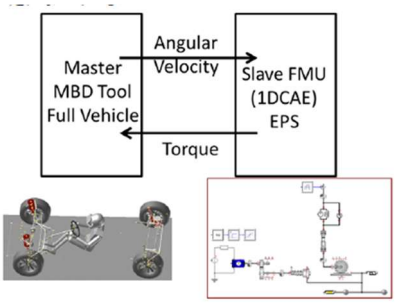


図 2.2.1 FMI を用いたツール連携の利用種別

2.2.3. 3DCAE 連携の実態

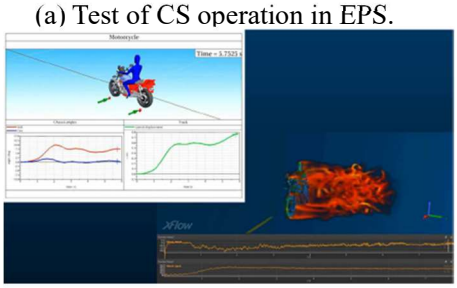
適用事例について図 2.2.2～図 2.2.5 に示します。ケース①では、EPS 開発における CS 運用の試験[3]や、CFD を用いた横風解析と回復制御ロジックとの連携[4]などが報告されています。前者では CS の主副、CS 制御ツールを介した接続を行い、実行時間への影響や誤差累積の様子などを検証しています。コシミュレーションの実現が容易になることが利点である一方、コミュニケーションステップサイズの策定に課題があるなどが報告されています。

ケース②では機構解析による車両挙動計算と燃料タンク内のスロッシング解析、及び車両挙動に及ぼす影響解析[5]や、連続鍛造時における剛性の変化に応じた器機の制御ロジックの開発に利用した事例[6]などが報告されている。本ケースでの FMU 利用にあたっては 3D ツールユーザにとって慣れ親しんだツール環境にて高度な制御モデルや運動モデルを利用できるため、活用に期待との意見も見られました。



3D MultiBody ↔ 1D CAE

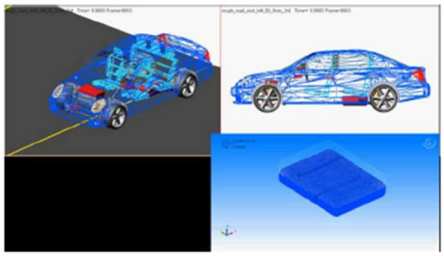
(a) Test of CS operation in EPS.



CFD → Multi Body

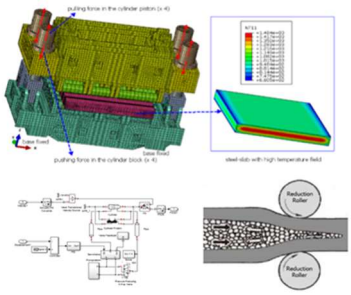
(b) Crosswind to MC and recovery control.

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



Multibody→(CS)→ CFD

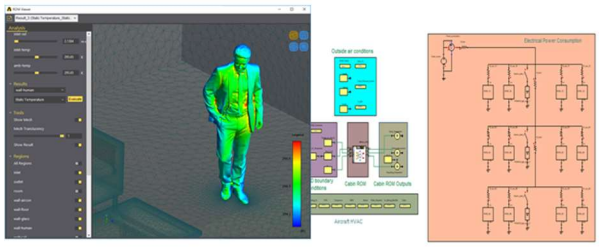
(a) Fuel sloshing on rough roads.



0D Sim → Structure

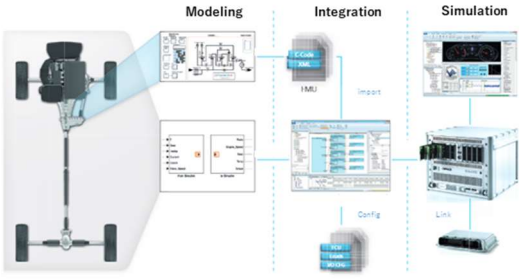
(b) Control for continuous forging.

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



CFD →(ME)→ 1D Sim
HVAC using SVD reduced model[7].

図 2.2.4　3D (reduced ME) with 1D simulation.



1D Sim →(CS)→ HIL
Verification of 8-speed AT-ECU.

図 2.2.5　1D with HIL simulation.

ケース③（図 2.2.4）及びケース④（図 2.2.5）においても、その活用のシーンが広まっています。縮約モデルは適切な 3D モデルの特性を高速に 1D シミュレーションにて活用でき、今後の性能向上が期待されています。また、リアルタイムでの実行は環境が整いつつあり、近年では対応ツールとハードウェアが拡大しており、モデルパラメータの検証などより現実的な運用へと移行していますが、CS を用いるケースと ME を用いるケースなどハードウェア環境により接続形式は多様であり、その影響については今後の検討課題です。

2.2.4.　3D CAE 連携の効果と課題

3D 解析エンジニアにとって、従来は他ツールとのコシミュレーションを実現するためにはユーザ定義関数（C 言語によるツール独自の関数など）を用いたプログラム開発が必要であり、バージョン管理やアップデート等の保守管理に大きなコストが費やされていました。FMI により標準化されたインタフェースを用いる事により高度な制御や運動モデルを境界条件として適用するなどが可能となると共に、新たなツール操作の取得時間も最低限とする効果が得られます。このため、流体や塑性変形など非線形性の強い物理モデルを用いた詳細な制御設計との連携が容易となり、3D シミュレーションの適用分野の拡大が期待されます。

また 1D システムシミュレーションエンジニアにとっては設計段階での 3D 詳細検証に活用し、設計をフロントローディングすることが期待できます。更に、SIL、 MIL、 HIL で一貫したモデルの活用などから IoT プラットフォームでの意思決定のための解決手段としてのモデル活用なども今後の展開の視野に入ってきます。

一方で 3D シミュレーションは年々そのモデル規模が拡大する傾向にあり、3D 計算時間が高コストでありネックであることは否めません。並列化や GPU の利用など高速化対策が必須であり、今後の技術開発が期待されます。また、FMI 2.0/CS の規格においては Roll Back（時間巻き戻し）への対応がオプション扱いであり[8][9]、複数のツール連携では問題が生じる可能性があります。Roll Back はスティフな系において当たり判定等のイベントが生じた際にその時刻に巻き戻す手法ですが、連続体計算用の CFD が本機能に対応しているかはツール次第であり、複雑な機構との連係には課題が残ります。

今回の調査では大規模 3D ツール連携での効果を確認することが出来ませんでしたが、複数の 3D ツールを連携した事例（マルチフィジックスやモデル結合など）や、モデル規模が大きい場合のデータ通信量によるオーバーヘッドなどの課題等に関しては今後の調査にて検討したい。

本節の参考文献

（1）経済産業省 MBD 研究会：「自動車開発におけるプラントモデル I/F ガイドライン解説書 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）広野、田中、松本、”EPS 開発におけるマルチボディ解析ツールと 1DCAE ツールの FMI 連成”、 自動車技術会 2016 春大会予稿集、pp2167-2170、 (2016)

（4）Dssault Systems チュートリアル

（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 セミナー

（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. C/C++, Python モデルと商用シミュレーションツールの連成

2.3.1. ユースケース

これまでの章で紹介されているように、FMI は多数の商用シミュレーションツールにおいて、連成シミュレーションのインタフェースの一つとして採用されています。連成させたいモデルを、FMI に対応している商用シミュレーションツールあるいはオープンソースのシミュレーションツールで開発していた場合は、他の章で紹介している手法で FMI による連成を検討すればよいですが、ここでは、連成させたいモデルが、C/C++言語や Python で構築されていた場合の連成方法を紹介します。

ユースケースとして以下の２つを取り上げます。

ユースケース１：自社製シミュレータを商用シミュレーションツールと連成させたい
ユースケース２：機械学習・深層学習モデルを商用シミュレーションツールと連成させたい

ユースケース 1 は、C/C++言語で開発された自社製品のシミュレータあるいはシミュレーションモデルを、FMI に対応した商用シミュレーションツールと連成させるケースです。FMI は C 言語の関数としてインタフェースが規定されていますので、C 言語への FFI（Foreign Function Interface）を持つプログラミング言語であれば、原理的には FMU との連成が可能ですが、ここではその詳細は立ち入らず、C/C++で開発された自社製シミュレータを商用シミュレーションツールと連成させるケースを考えます。

ユースケース２は、Python 言語で開発された機械学習・深層学習のモデルを、FMI に対応した商用シミュレーションツールと連成させるケースです。近年、TensorFlow（https://www.tensorflow.org）や PyTorch（https://pytorch.org/）に代表される機械学習・深層学習モデルを作成するオープンソースのライブラリを利用して作成されたモデルが、エンジニアリング活動の様々な局面で用いられています。そのような Python で構築されたモデルを、商用シミュレーションツールと連成させるユースケースを考えます。

2.3.2. ユースケース 1 の実現方法

ユースケース１を実現する方法は以下の２通りあります。

タイプ A：自社製シミュレータ側で FMI を実装し、商用シミュレーションツールへツールカップリング型の Co-Simulation FMU としてインポートする方法
タイプ B：自社製シミュレータ側に FMU インポート機能を追加して、商用シミュレーションツールからエクスポートした FMU を自社製シミュレータにインポートする方法

ぞれぞれのアーキテクチャを図 2.3.1 と図 2.3.2 に示します。

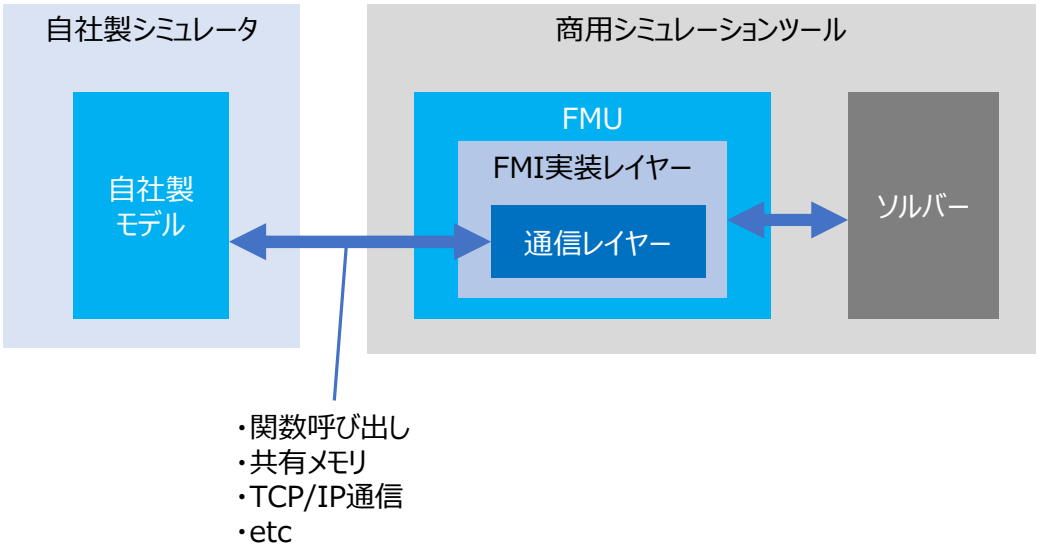


図 2.3.1 自社製シミュレータ側で FMI を実装する場合

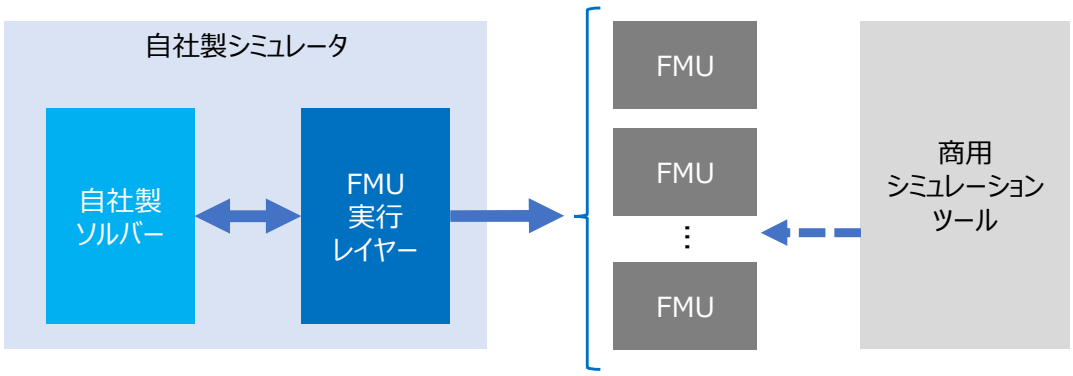


図 2.3.2 自社製シミュレータ側に FMU インポート機能を実装する場合

タイプ A の場合は、商用シミュレーションツールに FMU としてインポートさせるために、図 2.3-1 における FMI 実装レイヤーと通信レイヤーを実装する必要があります。FMI 実装レイヤーとは FMI 仕様書で定義されている C 言語関数を実装するレイヤーです。FMI として必要な関数はそれほど多くはないですが、以下のオープンソースソフトウェアを利用すると効率的に開発することができます。

名称：Reference-FMUs
URL：https://github.com/modelica/Reference-FMUs
ライセンス：2 条項 BSD ライセンス

また、図 2.3.1 における通信レイヤーとは、FMI の各関数が呼び出されたら、自社製シミュレータを呼び出すことでその仕様を実現するレイヤーです。これは、関数呼び出し、共有メモリ等によるプロセス間通信、あるいは TCP/IP 通信などの手段で実現することができます。このうち、TCP/IP 通信を利用する場合は、例えば以下のライブラリを利用すると高速な通信を実現できます。

名称：gRPC
URL：https://github.com/grpc/grpc
ライセンス：Apache ライセンス 2.0

タイプ B を実現するには、FMU 実行レイヤーを実装する必要があります。この場合は、手軽に利用できるオープンソースソフトウェアはありませんが、FMI の実行モデルをよく理解すれば実装することが可能です。FMI のバージョン 2.0 であれば、FMI 仕様書（Functional Mock-up Interface for Model Exchange and Co-Simulation）の 3.1 章および 4.1 章に実行モデルが記載されています。また、さきほど紹介した Reference-FMUs には FMU をインポートして実行するためのサンプルプロジェクト fmusim が含まれています。これらを参照すると FMU インポート機能を実装するために必要な内容を理解することが可能になります。

2.3.3. ユースケース 2 の実現方法

ユースケース 2 を実現する方法は以下の 2 通りあります。

タイプ C：Python 処理系および自社製 Python プログラムを組み込んだ FMU を実装し、商用シミュレーションツールにインポートする。

タイプ D：Python 環境に、FMPy を利用した FMU 実行機能を追加して、商用シミュレーションツールからエクスポートした FMU を Python 環境にインポートする。

ぞれぞれのアーキテクチャを図 2.3.3 と図 2.3.4 に示します。

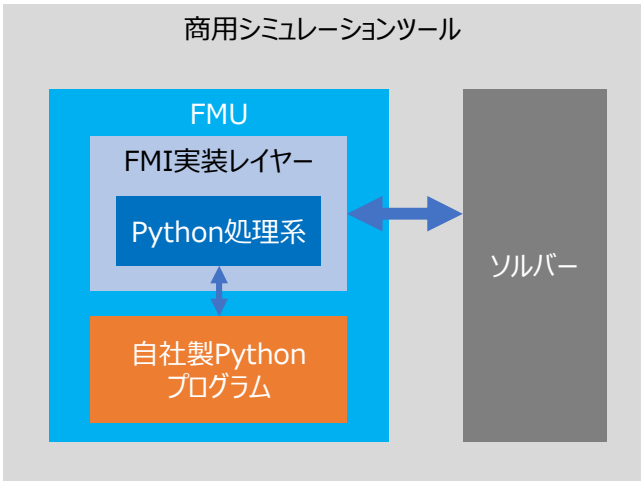


図 2.3.3 Python 処理系を組み込んだ自作 FMU を実装する場合

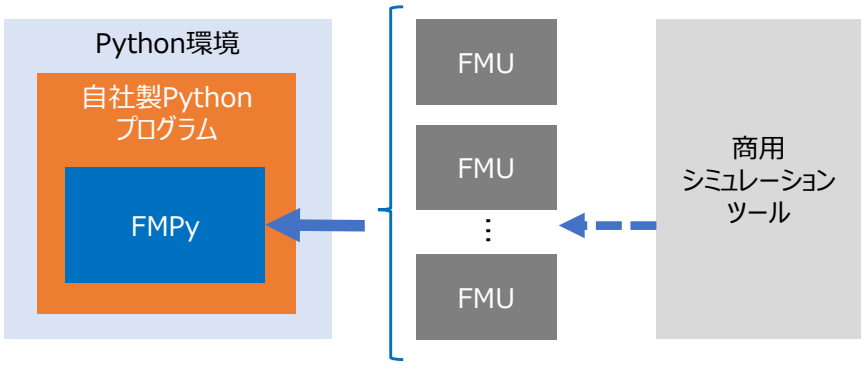


図 2.3.4 Python 環境に FMU をインポートする場合

タイプ C ですが、FMI 実装レイヤーを Reference FMUs などを利用して実装することはタイプ A と同じですが、タイプ C では Python 処理系を FMU に組み込みます。Python 処理系は他のアプリケーションに組み込む機能を持っているので、これを利用すると Python 実行機能をまるごと FMU に組み込むことができます。これにより FMU の各関数が呼ばれた際に自社製 Python プログラムを実行できるようになり、商用シミュレーションツール側からみると Python プログラムを FMU として動作させられるようになります。

しかし、注意点として、自社製 Python プログラムが、機械学習・深層学習ライブラリなど多数の外部ライブラリに依存している場合、それらも FMU に同梱する必要があります。依存ライブラリを全て FMU に同梱できること、および FMU として商用シミュレーションツールにインポートした場合にそれらが正しく動作することを別途検証する必要があり、ある程度規模が大きい Python プログラムを FMU に同梱することは現実的には採用が難しい場合が多いです。

タイプ D は、本書の 1.12 節でも紹介している FMPy を利用して、自社製 Python プログラムに FMU 実行機能を付与する方法です。

名称：FMPy
URL：https://github.com/CATIA-Systems/FMPy
ライセンス：2 条項 BSD ライセンス

FMPy は、1.12 節で紹介しているような GUI を利用して FMU を実行することができますが、FMU 実行機能だけを既存の Python プログラムに容易に組み込むことができます。これを利用すれば、自作の Python プログラムと FMU を組み合わせて動作させるプログラムを Python で実装することができるようになります。このとき、Python プログラム側は FMU である必要はないので、機械学習・深層学習モデルそのものであっても、FMU と連携させるようなプログラムを開発することが可能です。タイプ C よりも実現が容易なため、Python プログラムと FMU を連成させたい場合は、タイプ D により実現することを推奨します。

2.4. FMI に準拠した車載通信プロトコル(CAN)の実現

2.4.1. 進化する E/E アーキテクチャ開発が抱える課題

ソフトウェアがクルマの新しい価値を創出するようになり、ユーザのアプリケーションソフトウェアは大規模化、複雑化する一方です。SW の効率的な開発手法は、高度な車両開発にとって、欠かせない要件となりました。

E/E アーキテクチャの進化に伴い、多くのデバイスが搭載される傾向にあり、これらデバイスに搭載される SW 同士が相互に接続された状態での SW 検証が必須です。

各デバイスが揃い、相互接続できれば相互接続した SW 検証が行えますが、開発終盤まで課題検出が出来ず、大きな手戻りが発生します。

また、デバイス単体シミュレーション環境を利用すれば早期 SW 開発に着手できます。しかし、相互接続できず、複数のデバイスが連携したシステムレベルの検証が出来ません。このため、想定した複雑なシナリオに基づいた検証となり、品質の確保が難しくなってしまいます。

2.4.2. マルチデバイス用協調シミュレーション環境の概要

図 2.4.1 マルチデバイス用協調シミュレーション環境の構成

・マルチデバイス用協調シミュレーション環境の概要

[**マルチデバイス用協調シミュレーション環境**の構成] を図 2.4.1 に示します。

ルネサスは、実機無しでデバイス間を相互接続しソフトウェア開発を開始可能とするマルチデバイス用協調シミュレーション環境を提案しています。

・マスタツール(*1)経由で各 VPF を容易に接続するための FMU(*2)及び制御サンプルブロック
・FMU と VPF を接続するための通信スクリプト
・各実機の動作を模擬する VPF(*3),
・相互接続する VPF 上で動作する通信サンプル SW

を利用する事で、実機の入手前に相互接続したソフトウェア開発を開始出来ます。

VPF 上で開発したソフトウェアは実機でも動作するため、実機入手後に遅延なく、実機上でのソフト

ウェア開発・検証を実施出来ます。これにより、早期に相互接続された SW の開発が開始出来、早い段階での問題検出が可能となります。（マスタツール及び VPF は、御利用される方により入手が必要）

(*1) マスタツール 各 VPF の動作を調整し同期動作させるためのツール
(*2): Function Mockup Unit(FMU) 異なるツール間を接続するためのモデル IF である Function Mockup Interface（FMI）に従い、ツール間を接続するためのライブラリ。マスタツールからインポートする事で、マスタツール内で相互接続可能になる。
(*3): Virtual Platform(VPF) 実機同様の SW が実行可能な、SOC や MCU のシミュレータ。

2.4.3. CAN モデル同士の接続のための施策

[FMI で接続されたモデル間における CAN 通信の実現方法]を図 2.4.2 に示します。FMI では、どのような型やタイミングでデータ送受信が行えるかについて定義していますが、具体的な接続仕様については定義がありません。どのパラメータをどのように接続するかは、利用者任せになっており、FMI での接続仕様を定める必要があります。

このため、マルチデバイス用協調シミュレーション環境では、図 2.4.2 の FMI import Block 同士が接続するための CAN の接続仕様を策定しました。この仕様に準じた FMI を持ったモデル同士であれば、異なるツール間であっても接続可能になります。

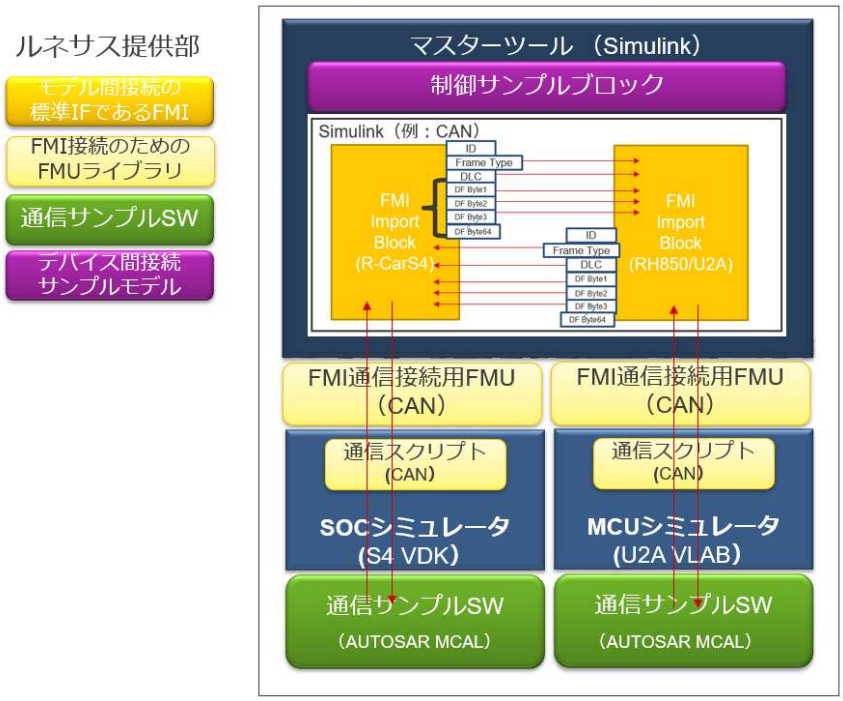


図 2.4.2 FMI で接続されたモデル間における CAN 通信の実現方法

2.4.4. マルチデバイス用協調シミュレーション環境の開発状況

今回、R-CarS4 及び RH850/U2A を CAN 接続したマルチデバイス用協調シミュレーション環境を紹介しました。

(Introduction of Virtual Platform Co-simulation | Renesas)に実際の動作を録画したデモビデオが公開されているので、ご参照下さい。

また､R-CarS4 及び R-CarV4H を Ethernet 接続したマルチデバイス用協調シミュレーション環境や、CAN や Ethernet のバスを模擬したモデルを Simulink 上に用意した環境の実証確認も行いました。

このように、FMI による通信プロトコル仕様の共通化（公開）により、異なるツール同士での接続が容易になり、大規模シミュレーション環境における、早期 SW 開発の実現により、SW の構造からシステム全体の構成を決定する SW ファーストの開発に貢献できる見込みを得ました。

第 3 章　チュートリアル

この章では、FMI を活用するための実践的な例題を用意しました。

3.1. チュートリアル 1 ：FMI を使ってみよう

3.1.1. サンプルモデルを準備しよう

自動車技術会ホームページの自動車制御とモデル部門委員会ページに用意したサンプルモデルを実際に取り込んでシミュレーションを実行してみましょう。モデルは、2018 年 5 月の自動車技術会春季大会にて、当 WG で講演した際のベンチマークモデルを使用します。[1]

ファイル名：Sample_3p1.zip　：ZIP ファイルには以下 4 個のファイルが格納されています。

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

今回の例題は、簡単な回転ばねダンパモデルを分割したもので、FMU は下記にて作成されています。

Version	: FMI 2.0
Type	: ME および CS
Compiler	: Visual C++ 64bit
OS	: Windows 64bit

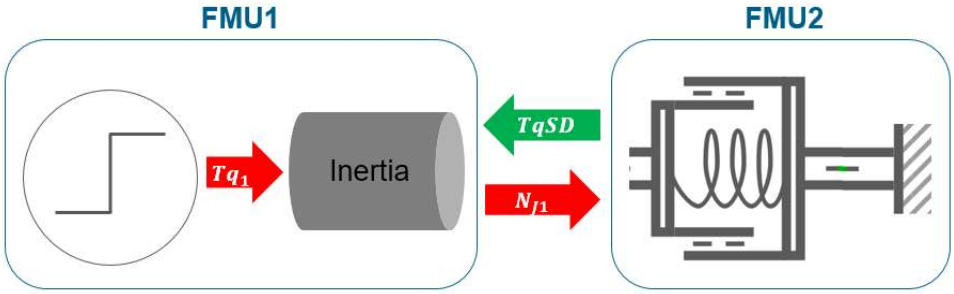


図 3.1.1　モデルの構成

まずは、Import に対応したシミュレーションツールを用意する必要があります。本編 2.9 節で紹介した FMI ホームページにてお手持ちのツールの FMU Import 対応状況を確認してください。

ツールが用意できれば、まずは FMU を取り込みますが、その手順は使用ツールよって異なりますので、ここでは割愛します。

うまく取り込めたでしょうか？

以降は、Simcenter Amesim での実行例をご紹介いたします。

3.1.2. Model Exchange でシミュレーションしよう

3.1.2.1. 標準パラメータで実行

次に、取り込んだ FMU を接続していきます。まずは、FMU1、FMU2 ともに Model Exchange を使います。それぞれの FMU には 1 入力、1 出力のみである為、双方の FMU の出力と入力をつなげば、問題なく図 3.1.2 のように接続できるはずです。

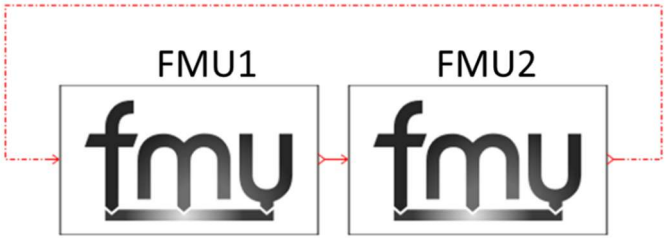


図 3.1.2　FMU の接続

次に FMU のパラメータを設定します。

図 3.1.3a、図 3.1.3b のパラメータ”path to the unzipped FMU root”（ツールにより異なります）は、解凍した FMU の dll 格納場所を指定します。シミュレーション実行時にはこのアドレスにある dll を呼ぶのでパスが重要です。

#から始まる値は積分計算を行う状態変数の初期値です。変更可能ですが、まずはこのままの設定とします。#などの変数は使用するツールにより表示が異なるのでご注意ください。

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	

図 3.1.3a　FMU1 (Model Exchange) パラメータ

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	

図 3.1.3b　FMU2 (Model Exchange) パラメータ

早速、FMU に設定されている Default のパラメータセットでシミュレーションを実行して結果を確認してみましょう。回転数 Nj1 とトルク TqSD は図 3.1.4 のような結果となったでしょうか？

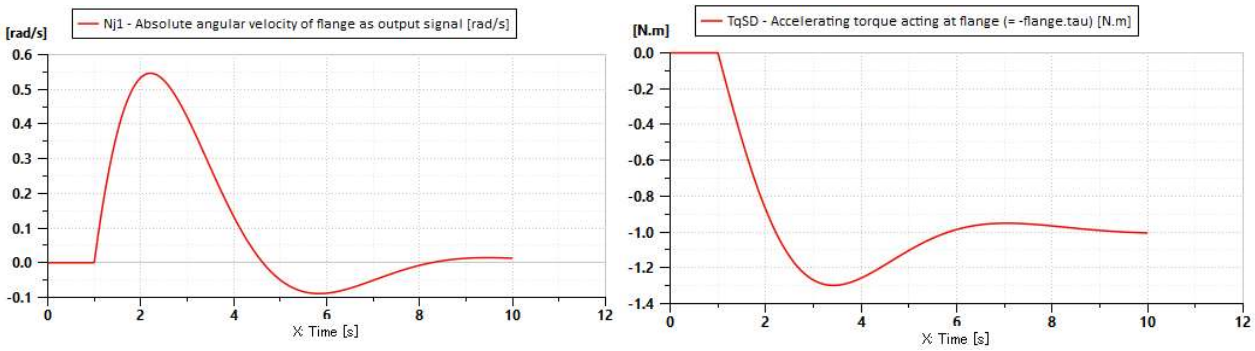


図 3.1.4　シミュレーション結果（Model Exchange）

3.1.2.2. パラメータを変更して実行

次に、パラメータを変更してみましょう。共振周波数を上げ、ダンピングを低く設定します。

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	

図 3.1.5　FMU1（Model Exchange）変更前パラメータ

- バネ定数（spring.c - Spring constant）：
  現在の”1 [N･m/rad]”から 10 倍の”10 [N･m/rad]”へ変更
- ダンピング（damper.d - Damping constant）：
  現在の”1 [N･m･s/rad]”から 1/10 の”0.1 [N･m･s/rad]” へ変更

変更後シミュレーションを実行して、結果を確認してください。

回転数 Nj1 とトルク TqSD は図 3.1.6 のような結果となったでしょうか？

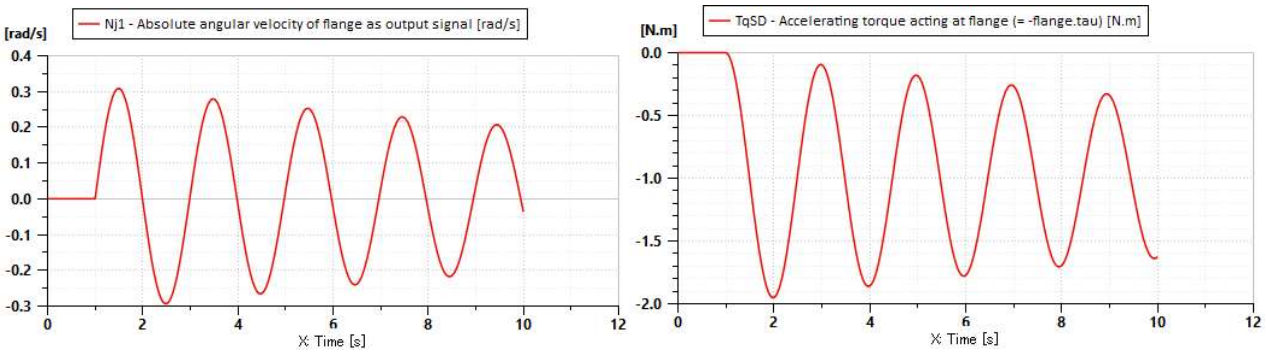


図 3.1.6　シミュレーション結果（ME：パラメータ変更後）

3.1.3. Co-Simulation でシミュレーションしよう

3.1.3.1. 標準パラメータで実行

次に、Co-Simulation の FMU を取り込み、モデルを接続しパラメータを確認してみましょう。

パラメータの内容は、Model Exchange 時とほぼ同じですが、下記の 2 つが各 FMU で増えています。

- 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	

図 3.1.7 a　FMU1 (Co-Simulation) パラメータ

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	

図 3.1.7b　FMU2 (Co-Simulation) パラメータ

これらは、Co-Simulation を行う際の通信間隔を設定するパラメータです。ここで表記されるパラメータの名前は、取り込み側のツールにより異なることがあります。今回は、”0.001 [s]”が通信間隔として設定されており、この間隔ごとに各 FMU への入力がやりとりされます。

それでは、FMU に設定されている Default のパラメータセットでシミュレーションを実行して結果を確認してみましょう。

回転数 Nj1 とトルク TqSD は図 3.1.8 のような結果となったでしょうか？

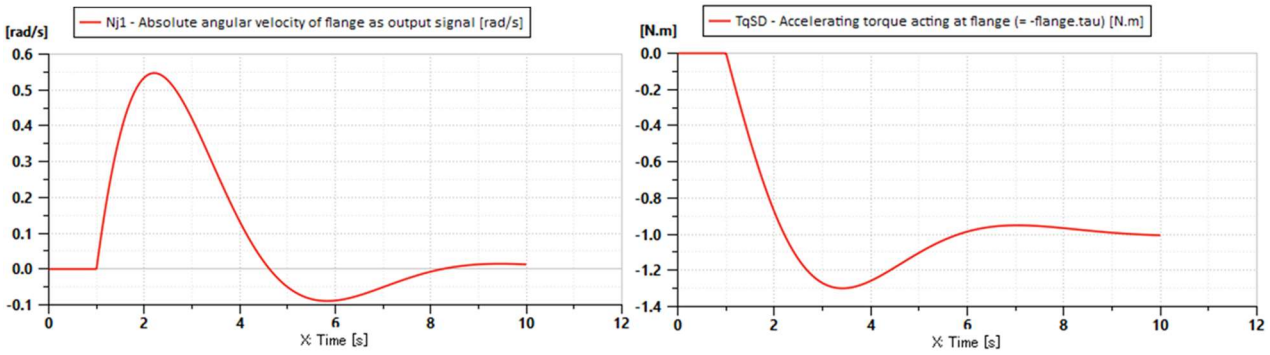


図 3.1.8　シミュレーション結果　(Co-Simulation)

Model Exchange 同様のシミュレーション結果がみられます。しかし、Co-Simulation を行う際には、通信間隔間では FMU の入出力は一定値として扱われ、Step Delay が生じます（本編 2.4.2 節参照）。この例では、0.001[s]という細かな通信間隔なので認識は困難です。出力結果のサンプリング設定を細かくして（0.0001[s]など）グラフを拡大してみましょう。図 3.1.9 のように通信間隔の 0.001[s]毎に結果が離散的に変化していることがわかります。

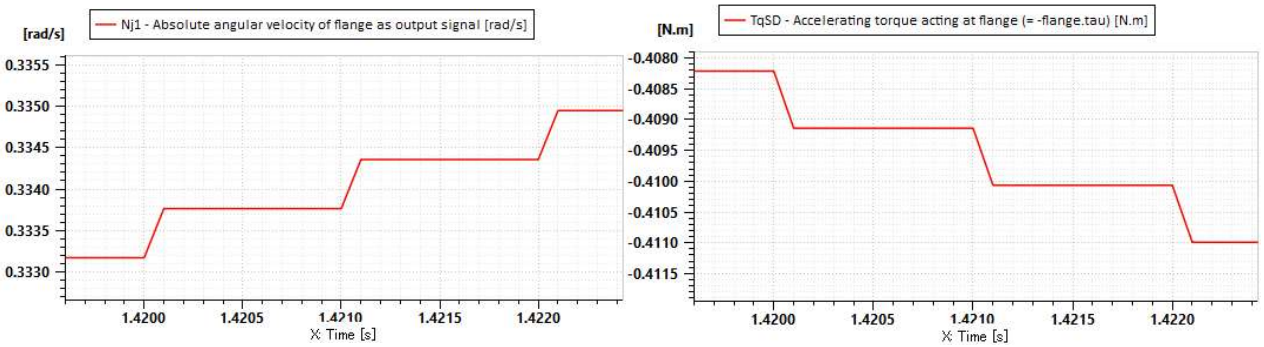


図 3.1.9　シミュレーション結果　(図 3.1.9 拡大)

3.1.3.2. パラメータを変更して実行

次に、パラメータを変更してみましょう。FMU1、FMU2 の通信間隔を粗く設定します。

- 通信間隔（co-simulation step size）：現在の”0.001 [s]”から 100 倍の”0.1 [s]”へ変更

通信間隔が粗くなり、離散的な結果が見えやすくなっています。また、通信間隔 0.001[s]の結果と比較すると、結果が異なっていることが見て取れます。Co-Simulation では、適切な通信間隔の設定が重要であることがわかります。詳細については 4 章で述べますので、そちらをご参照ください。

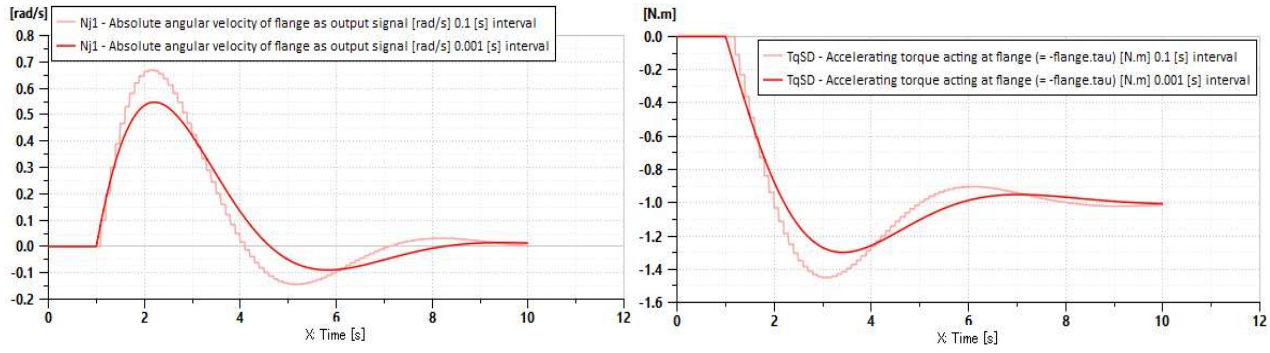


図 3.1.10　シミュレーション結果（CS：パラメータ変更前後の比較）

3.2. チュートリアル２：FMI を作ってみよう

3.2.1. サンプルモデルを使って FMU を生成してみよう

サンプルモデルは自動車技術会ホームページの自動車制御とモデル部門委員会ページに格納しています。モデルは、2018 年 5 月の自動車技術会春季大会にて、当 WG で講演した際のベンチマークモデルを使用します。[1]

ファイル名：Sample_3p2.zip　　：ZIP ファイルには以下のファイルが格納されています。

表 3.2.1　サンプルモデルの内容

	Model Exchange のオリジナルモデル		Co-Simulation のオリジナルモデル	
作成ツール	FMU1 用	FMU2 用	FMU1 用	FMU2 用
Amsim	J1_Amesim_ME.ame	SD1_Amesim_ME.ame	J1_Amesim_CS.ame	SD1_Amesim_CS.ame
Simplorer	J1_SD1_Simplorer_ME_CS.aedt　(1 つのモデル内に 4 種類全てが収納されています)			
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	—	—

表 3.2.1 に対応するツールをお持ちであれば FMU を生成してみましょう。なお、FMU の生成方法は、使用するツールにより手順が異なりますので、詳細はツールのマニュアルをご参照下さい。

ここでは、Simcenter Amesim を用いた FMU の生成例をご紹介します。

まず、Co-Simulation のオリジナルモデルである FMU1 用の J1_Amesim_CS.ame を開いてください。図 3.2.1 の構成となりますが、このモデルはすでに FMI インターフェースブロックが挿入されており、FMU を生成する準備ができています。

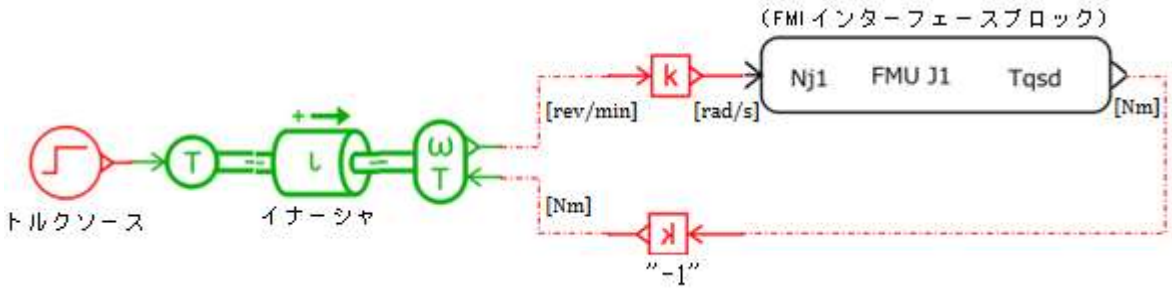


図 3.2.1　FMU1 出力用 J1_Amesim_CS.ame (J1 モデル)の構成

FMU の生成にあたり、入出力の単位系と信号の正負の向きに注意する必要があります。

モデル交換のルールは、経産省ガイドラインに基づいて決定しています（本編 4.1.1 節参照）。モデル間の受け渡しの単位は SI 組立単位系を採用するため、回転数は[rad/s]、トルクは[Nm]としています。

Amesim では回転数センサが[rev/min]を出力するので、”k”で単位変換して Nj1 を出力し FMU2 側へ出力します。

信号の正負の向きについては、エネルギソースからエネルギシンクへ流れる方向がエネルギの正の向きとなります。このモデルでは、トルクソース→イナーシャ→FMU2 側のバネ＋ダンパへの流れが正方

向にあたります。FMU2 側から流入するトルクは、エネルギ伝達と逆方向に働くので、イナーシャが正方向の回転となる場合には、回転数を低下させるため負方向のトルクが返ってくることになります。

Amesim のモデルでは、このトルクを正方向と定義しているため、”k”で－1 のゲインをかけて、正負の向きを反転するモデルにしています。この事例のトルクに限らず、物理量の正負の扱いはツールやモデルの作り方によって異なりますので注意が必要です。

モデルを FMU に出力するには、Interface > FMU Export Assistant を起動してください。図 3.2.2 のウィザードのメニューに従って FMU を生成します。

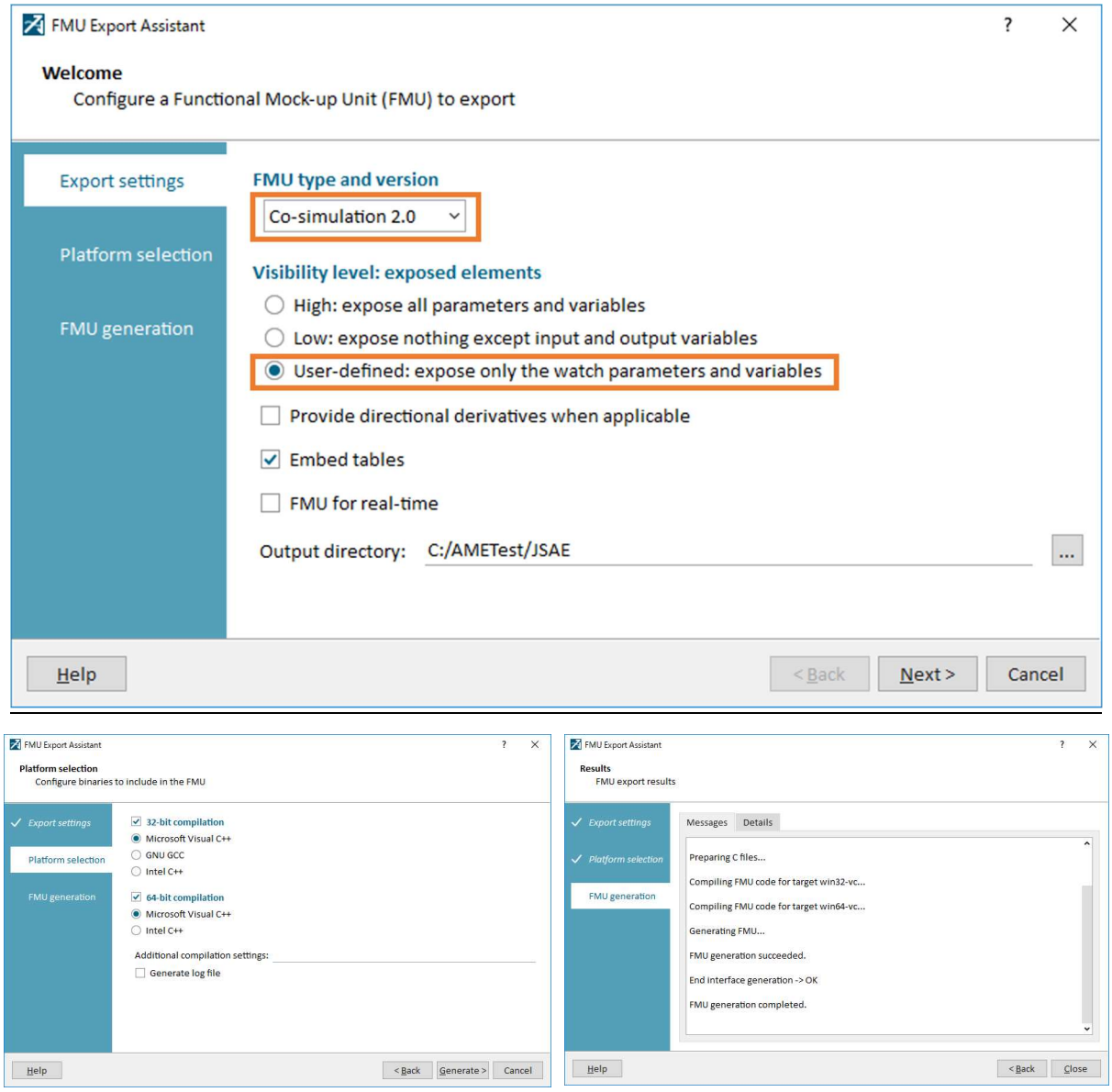


図 3.2.2　FMU Export Assistant

今回は FMI 2.0 の Co-simulation を行いますので、該当する”FMU type and version”を選択します。

次にパラメータの表示レベルを決定するため”Visibility level: exposed elements”を設定しますが、”User-defined: expose only the watch parameters and variables”を選択すると、FMU に表示するパラメータや変数を任意に設定できるため、モデル流通後の FMU を利用する側にとって使い易いモデルと言えます。具体的な事例は、3.3 節で説明します。

その後、使用するコンパイラを選択して FMU を生成します。

FMU が生成できたら、再び取込んでシミュレーションを実行してみましょう。

3.2.2. 必要なパラメータのみ表示しよう

FMU 生成時に、基となるモデルの作り方によっては、シミュレーション結果に影響を与えない不要なパラメータが表示されることがあります。不要なパラメータや変数が表示されると、FMU を利用する側の混乱を招くことになります。

ここでは、3.1.2 のサンプルモデルで使用した Model Exchange の FMU2 を例に説明します。図 3.2.3 のパラメータ”speed.f._crit – if exact=false、 critical frequency of filter to filter input s…”は変更可能となって見えていますが、実際にはこのパラメータを変更してもシミュレーション結果に影響を与えません。何故このようなことが起こるのでしょうか。

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	

図 3.2.3　FMU2 (Model Exchange) パラメータ

FMU2 を生成する前のオリジナルモデルは Modelica で作成しており、図 3.2.4 の構成となっています。

対応するツールをお持ちであれば SD1_Dymola_ME.mo（SD1 モデル）を開いてみましょう。

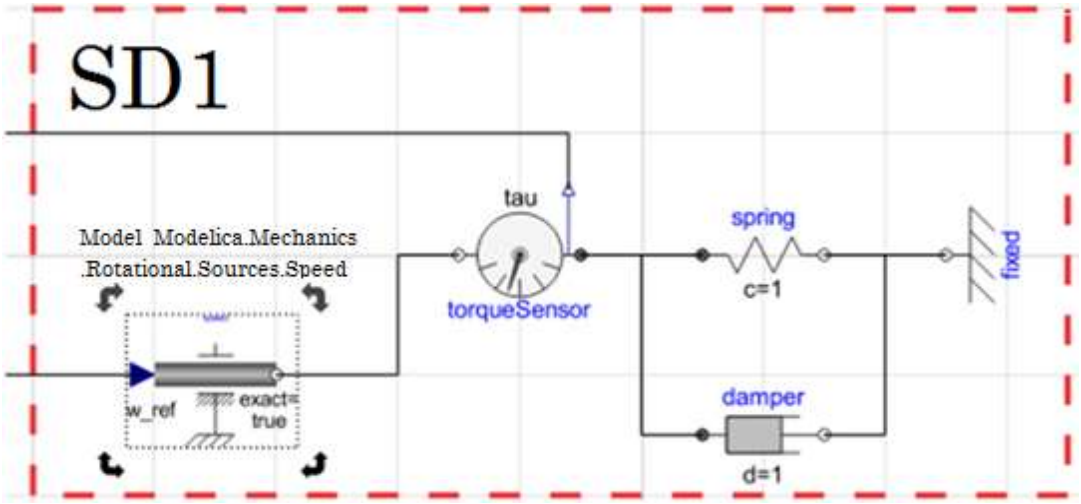
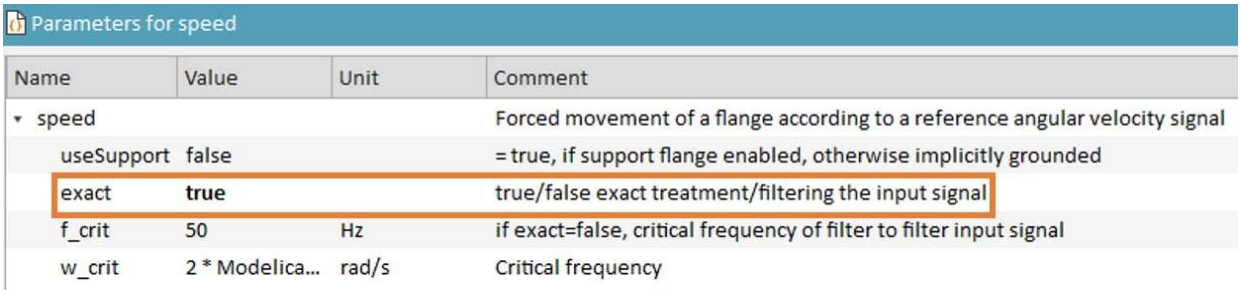


図 3.2.4　オリジナルの Modelica (SD1 モデル)の構成

このモデルでは“Model Modelica.Mechanics.Rotational.Sources.Speed“を使っており、このブロックのパラメータが図 3.2.5 になります。


Parameters for speed

Name	Value	Unit	Comment
speed			Forced movement of a flange according to a reference angular velocity signal
useSupport	false		= true, if support flange enabled, otherwise implicitly grounded
exact	true		true/false exact treatment/filtering the input signal
f_crit	50	Hz	if exact=false, critical frequency of filter to filter input signal
w_crit	2 * Modelica...	rad/s	Critical frequency

図 3.2.5 （SD1 モデル）speed のパラメータ

パラメータに”f_crit”がありますが、”exact”が false のときのみに有効になります。FMU 生成時に”exact”を”true”で設定すると、FMU 生成後は変更ができません。

したがって、図 3.2.5 で”speed.f._crit – if exact=false、 critical frequency of filter to filter input s…”の値を変更しても無効となります。

そもそも FMU 生成する際に”f_crit”を変更可能なパラメータとして扱ったために起こる現象なので、必要なパラメータや変数のみ表示できるように設定し、変更の必要がないパラメータは隠蔽することが、利用者の混乱を防ぐことに繋がります。

3.3. チュートリアル３：当 WG ベンチマークモデルの例

2017 年 10 月の自動車技術会 公開委員会「FMI (Functional Mockup Interface)によるモデル接続講習会」及び 2018 年 5 月の 2nd Japanese Modelica Conference で講演した当 WG のベンチマークモデル[2]を例題にチュートリアルを進めていきます。

3.3.1. サンプルモデルの説明

この車両モデルはドライバ、パワートレイン、ドライブライン、シャシ、ステアリングを 5 つの FMU で構成して Co-Simulation を行います。概略は図 3.3.1 を確認して下さい。

OS は Windows64bit 版、FMI2.0 のみに対応していますのでご注意ください。

サンプルモデルは自動車技術会ホームページの自動車制御とモデル部門委員会ページよりダウンロードできます。

ファイル名：Sample_6p2.zip　：ZIP ファイルには以下 7 個のファイルが格納されています。

Driver	: Driver_FMU_Export_Rev16.fmu.fmu
Power Train	: Powertrain_FMU_export_Rev16.fmu
Drive Line	: DS_Simulink.fmu
Chassis	: FMI_20_Blackbox_Euler_Test_FlatPad.fmu
Steering	: SteeringMechanismRackPinionInMetric.fmu
サブシステム I/F 定義書	: Sample_6p2_サブシステム I/F 定義書.xlsx
シミュレーション結果	: Sample_6p2_Result.txt

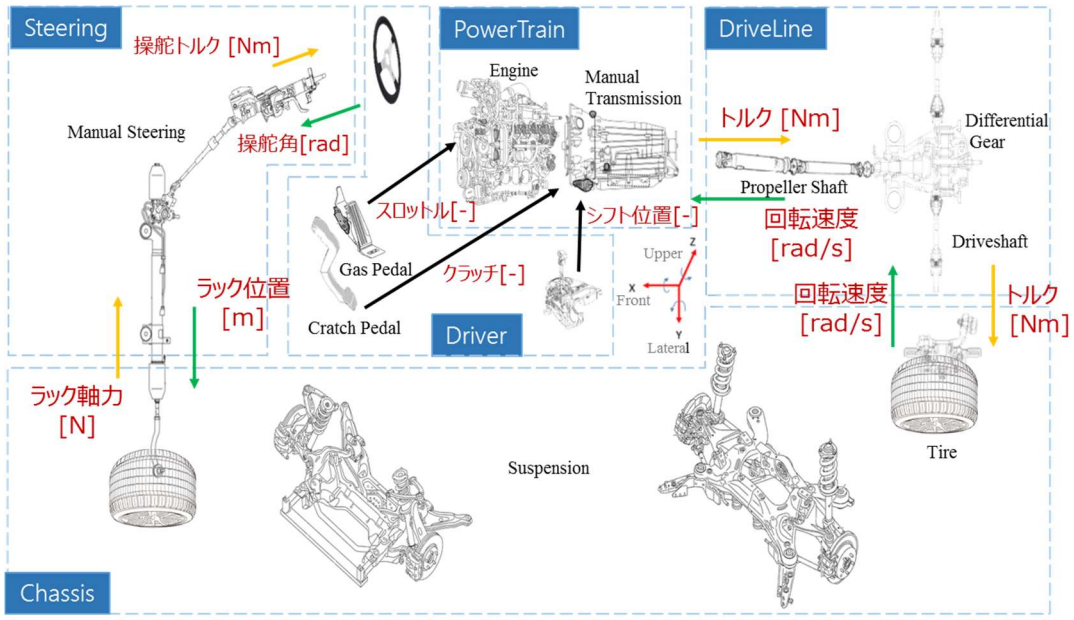


図 3.3.1　ベンチマークモデルの概要

まずは、使用するシミュレーションツールへ 5 つの FMU を取り込みます。取り込み方法は、メニュー画面で指定する場合や Explorer からドラッグ&ドロップなどツールによって異なりますので、マニュアルを参照して下さい。なお、各 FMU の内容について知りたい場合は、ZIP ファイルに同梱したサブシステム I/F 定義書（Sample_6p2_サブシステム I/F 定義書.xlsx）に概要を記載しています。

3.3.2. モデルの接続

次に、各 FMU を接続します。表 3.3.1 を参考に接続して下さい。緑色のハッチングはモニタ用出力のため接続には使用しません。図 3.3.2 はモデル接続例です。各 FMU の左側端子は入力、右側端子は出力になっています。使用するツールにより接続時の体裁は異なるためご注意ください。

表 3.3.1　入出力一覧表

FMU番号	端子番号		端子番号	入力端子			FMU番号	FMU名称	出力端子			端子番号
							1	driver	Clutch_Action@expseu_	[-]	クラッチ	1
									Engine_load@expseu	[-]	スロットル開度	2
									Gear_position@expseu	[-]	シフト位置	3
									Steerin_Angle@expseu	[rad]	操舵角	4
3	1	⇒	1	Npropshaft@expseu	[rad/s]	Prop/Shaft回転速度	2	Power Train	Neng@expseu	[rad/s]	エンジン回転速度	1
1	3	⇒	2	Gear_position@expseu	[-]	シフト位置			Nclutch@expseu	[rad/s]	クラッチ側回転速度	2
1	2	⇒	3	Engine_load@expseu_	[-]	スロットル開度			Tpropshaft@expseu_	[Nm]	Prop/Shaftトルク	3
1	1	⇒	4	Clutch_Action@expseu	[-]	クラッチ						
2	3	⇒	1	Tq_TM	[Nm]	Prop/Shaftトルク	3	Drive Line	N_Propshaft	[rad/s]	Prop/Shaft回転速度	1
4	7	⇒	2	N_RL	[rad/s]	後左輪D/Shaft回転速度			Tq_RL	[Nm]	後左輪D/Shaftトルク	2
4	5	⇒	3	N_RR	[rad/s]	後右輪D/Shaft回転速度			Tq_RR	[Nm]	後右輪D/Shaftトルク	3
5	2	⇒	1	steeringTranslation	[m]	ラック位置	4	Chassis	position_x	[m]	車両重心位置X方向	1
固定値[0]		⇒	2	FrontRightWheelTau	[N]	前右輪D/Shaftトルク			position_y	[m]	車両重心位置Y方向	2
3	3	⇒	3	RearRightWheelTau	[N]	後右輪D/Shaftトルク			position_z	[m]	車両重心位置Z方向	3
固定値[0]		⇒	4	FrontLeftWheelTau	[N]	前左輪D/Shaftトルク			FrontRightWheel_w	[rad/s]	前右輪回転速度	4
3	2	⇒	5	RearLeftWheelTau	[N]	後左輪D/Shaftトルク			RearRightWheel_w	[rad/s]	後右輪回転速度	5
									FrontLeftWheel_w	[rad/s]	前左輪回転速度	6
									RearLeftWheel_w	[rad/s]	後左輪回転速度	7
									steering_force	[N]	ラック軸力	8
1	4	⇒	1	steerAngle	[rad]	操舵角	5	Steering	steerTorque	[Nm]	操舵トルク	1
4	8	⇒	2	rackForce	[N]	ラック軸力			rackPosition	[m]	ラック位置	2

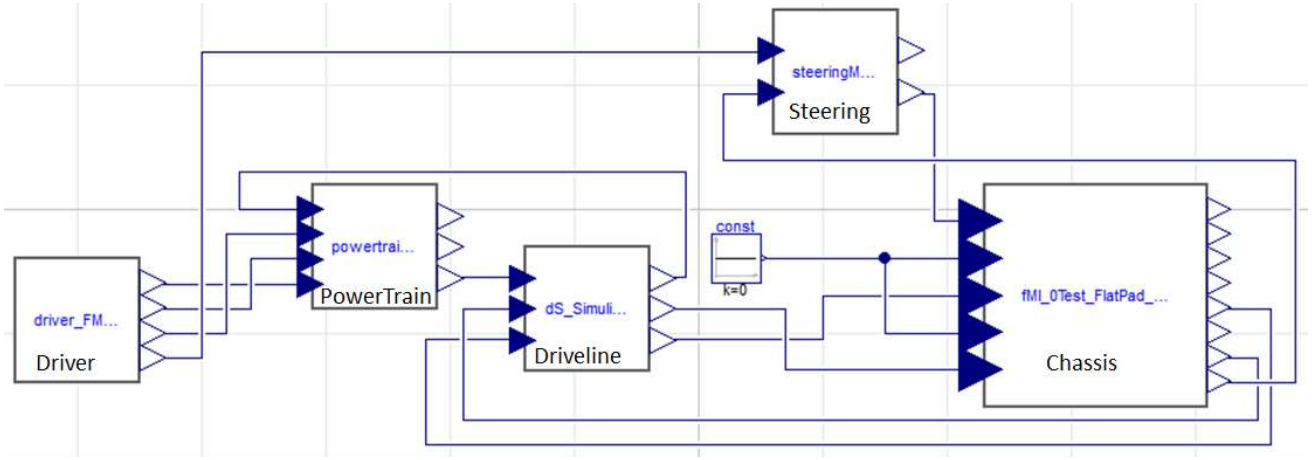


図 3.3.2　モデル接続例

3.3.3. ドライバモデルの説明

図 3.3.3 に示したように Open loop の単純な入力操作としています。マニュアルトランスミッション仕様のため、クラッチ操作を必要とし、停止から 5 速ギヤまで順次変速していきます。

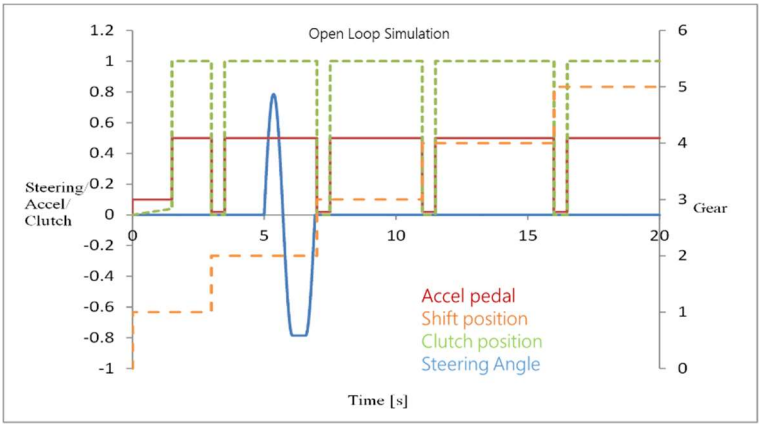


図 3.3.3　ドライバモデルの動作

3.3.4. FMU パラメータの設定

デフォルトのパラメータでは、正方向（左転舵）のステアリング操作に対してタイヤが右に切れてしまうため、設定を変更する必要があります。図 3.3.4 はステアリングモデルの概要です。ドライバモデルからの入力”steerAngle”に対して、”rackPosition”の移動方向を反転させるため、赤枠で示した”gs1”ブロックのパラメータを変更します。

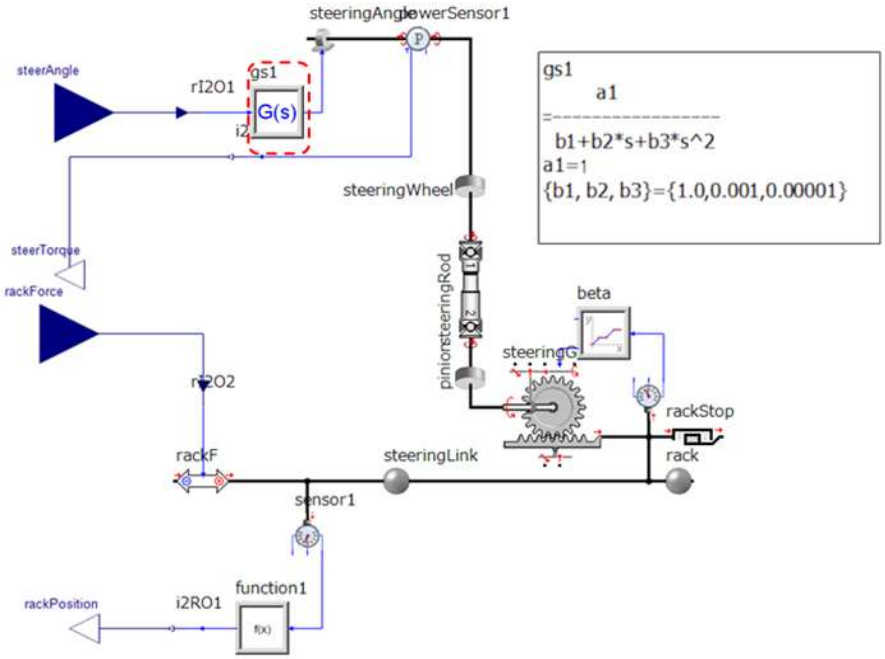


図 3.3.4　ステアリングモデルの概要

SimulationX の例では、図 3.3.5 のように Parameters のタブで設定します。

- 2 次伝達関数の分子（gs1.A[1]）：現在の ”1 [-]” から ”－1[-]” へ変更

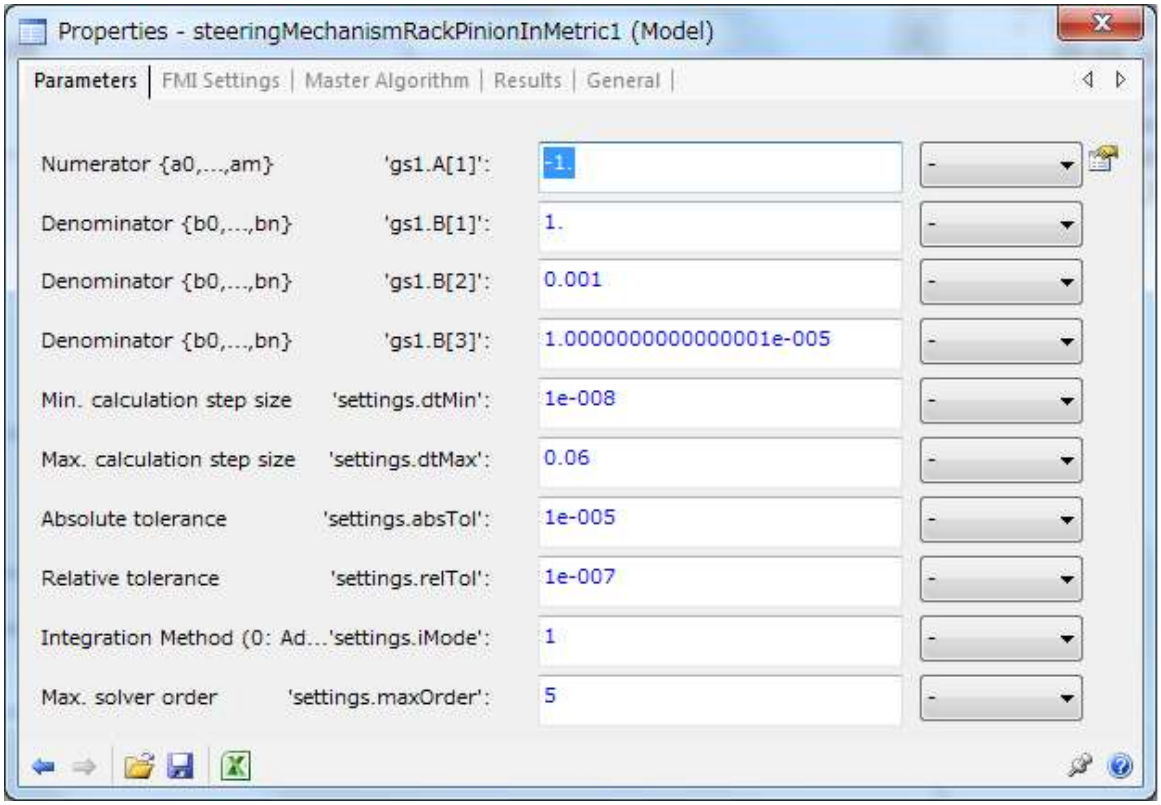


図 3.3.5　ステアリングのパラメータ設定（SimulationX での例）

3.3.5. マスタツールのシミュレーション設定

ドライバモデルで一連の操作が完了する時間の 20 秒をシミュレーション時間として設定します。
ソルバは固定ステップの”ODE1”相当とします。
最少ステップサイズは、5 つの FMU で、最も細かい計算間隔に合わせて”5e-5 [s]”とします。

Start Time	: 0 [s]
Stop Time	: 20 [s]
Solver、 Step Sizes and Tolerances	: Fixed step Solver（固定ステップ）
Integration method	: Euler Forward （ODE1 相当）
Min. Calculation Step Size	: 0.00005 （または 5e-5）[s]
Min. Output Step Size	: 0.01 [s] ・・・出力結果のサンプル時間なので任意

SimulationX の例では SimulationControl で設定します。

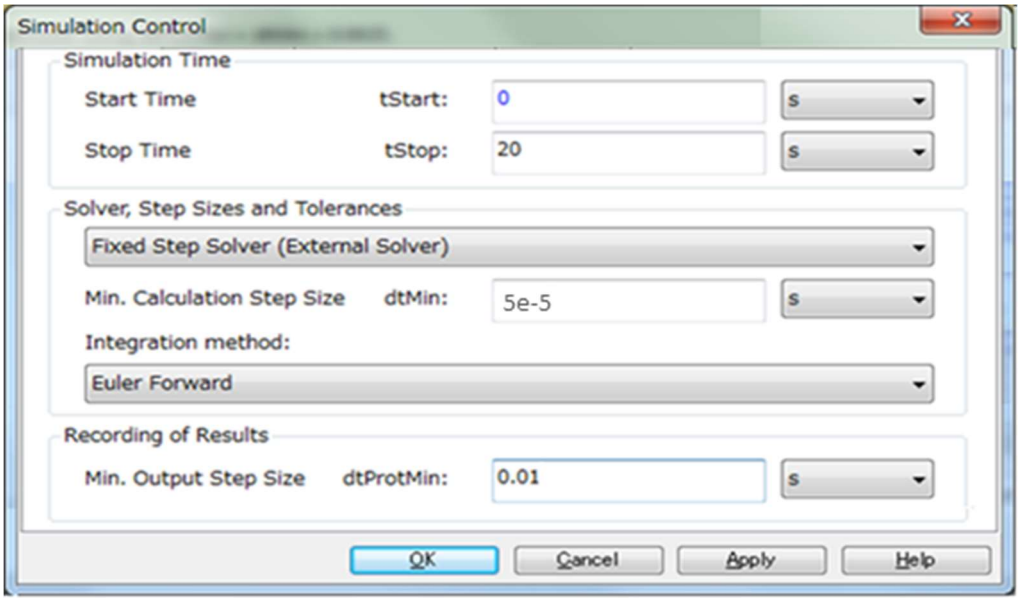


図 3.3.6　シミュレーション設定（SimulationX での例）

この例題において、マスタツール上は FMU のみで構成される Co-Simulation のため、ソルバの違いで大きな差は生じませんが、一般的な使われ方ではマスタツール上のモデルを計算し、さらに取り込んだ FMU との間で信号を受け渡すので、ソルバ、ステップサイズは慎重に設定する必要があります。

3.3.6. Communication Step Size の設定

Co-Simulation では FMU 毎に入力信号を受け取る時間間隔設定が必要となります。今回のモデルでは表 3.3.2 に従って全ての FMU に設定を行います。

表 3.3.2　各 FMU の Communication Step Size 設定

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

SimulationX の例では Master Algorithm のタブで設定します。各 FMU を同様に設定して下さい。

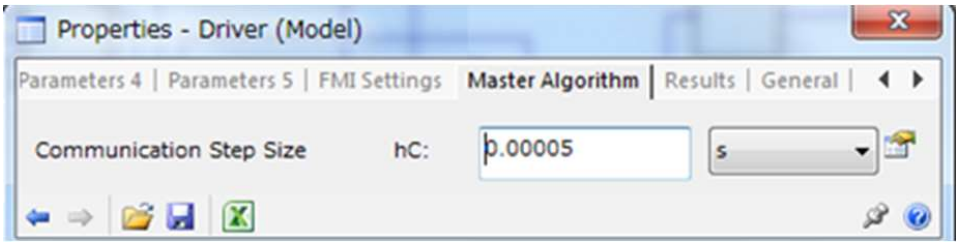


図 3.3.7　Communication Step Size の設定（SimulationX での例）

なお、Communication Step Size はツールによって名称が異なるため、代表例を紹介します。

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. シミュレーション結果：

各 FMU からの出力信号をいくつか掲載しました。同じような結果となったでしょうか？

詳細は Sample_6p2_Result.txt にシミュレーション結果を保存しているため参考にして下さい。

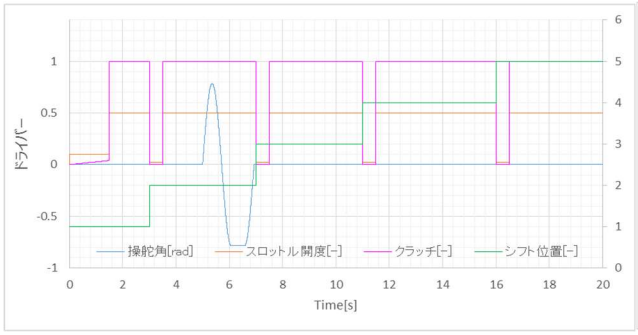


図 3.3.8 a　FMU1（出力）Driver

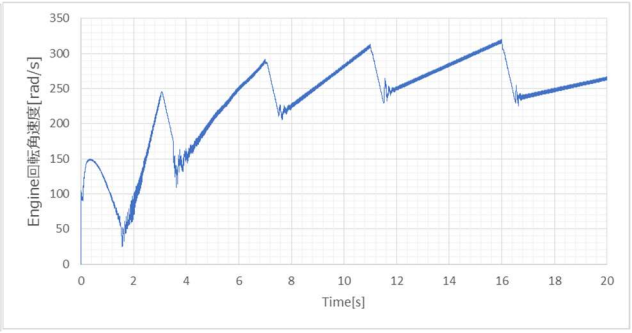


図 3.3.8b　FMU2（出力）エンジン回転速度

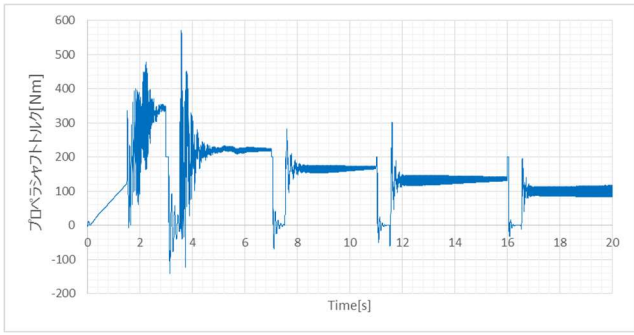


図 3.3.8c　FMU2（出力）Prop/Shaft トルク

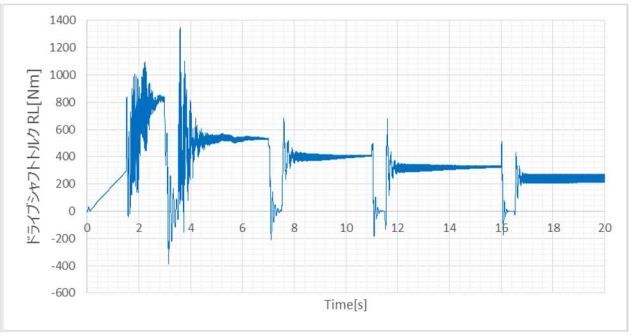


図 3.3.8d　FMU3（出力）後左軸 D/Shaft トルク

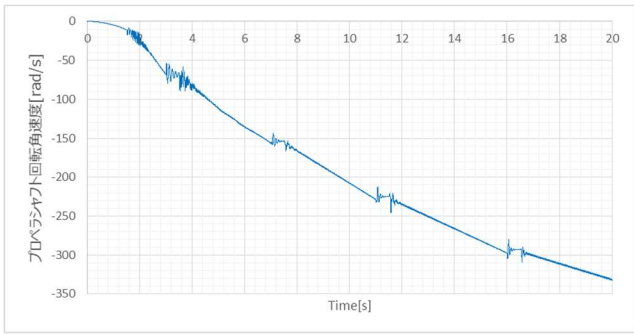


図 3.3.8e　FMU3（出力）Prop/Shaft 回転速度

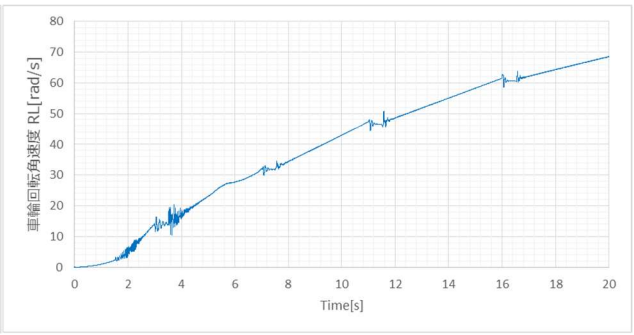


図 3.3.8f　FMU4（出力）後左軸 D/Shaft 回転速度

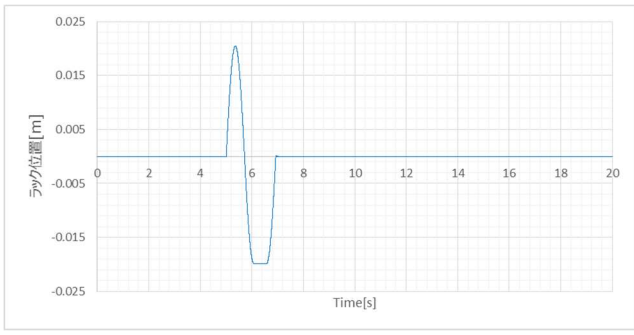


図 3.3.8g　FMU4（出力）ラック位置

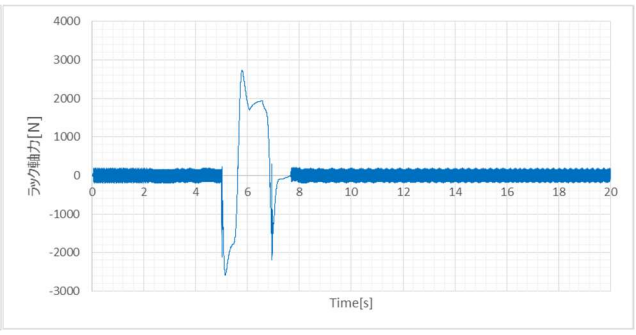


図 3.3.8h　FMU5（出力）ラック軸力

3.4. チュートリアル４：経産省ガイドラインに準拠したベンチマークモデルの例

2017 年 3 月に経済産業省の自動車産業におけるモデル利用のあり方に関する研究会より、ガイドラインが示され準拠モデル Ver.1.0（準拠モデルは Simulink ベースのモデル）が同時に公開されました。詳細は JAMBE ホームページで確認して下さい。

3.4.1. サンプルモデルの説明

ここからは、2018 年 5 月の自動車技術会春季大会で講演した当 WG のベンチマークモデルを例題にチュートリアルを進めていきます。[3]

経産省ガイドライン準拠モデルをベースに 7 つの FMU を作成しました。

Driver model（Split1）、Vehicle model については 6 つ（Split2～Split7）に分割しています。

サンプルモデルは自動車技術会ホームページの自動車制御とモデル部門委員会ページよりダウンロードできます。FMI2.0 Windows 64bit CS のみに対応していますのでご注意ください。

ファイル名：Sample_6p3.zip　　：ZIP ファイルには以下 9 個のファイルが格納されています。
FMU　：Driver.fmu、ENG_CNT.fmu、TM_CNT.fmu、PWT_PNT.fmu、CHA_PNT.fmu、
　　ALT_CNT.fmu、ELEC_PNT.fmu
JC08 モードでの目標車速の時系列データ　：jc08.csv
シミュレーション結果　　：Sample_6p3.txt

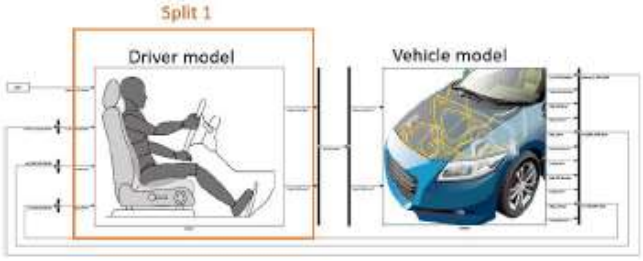


図 3.4.1　経業省ガイドライン準拠モデル 最上位階層の構成

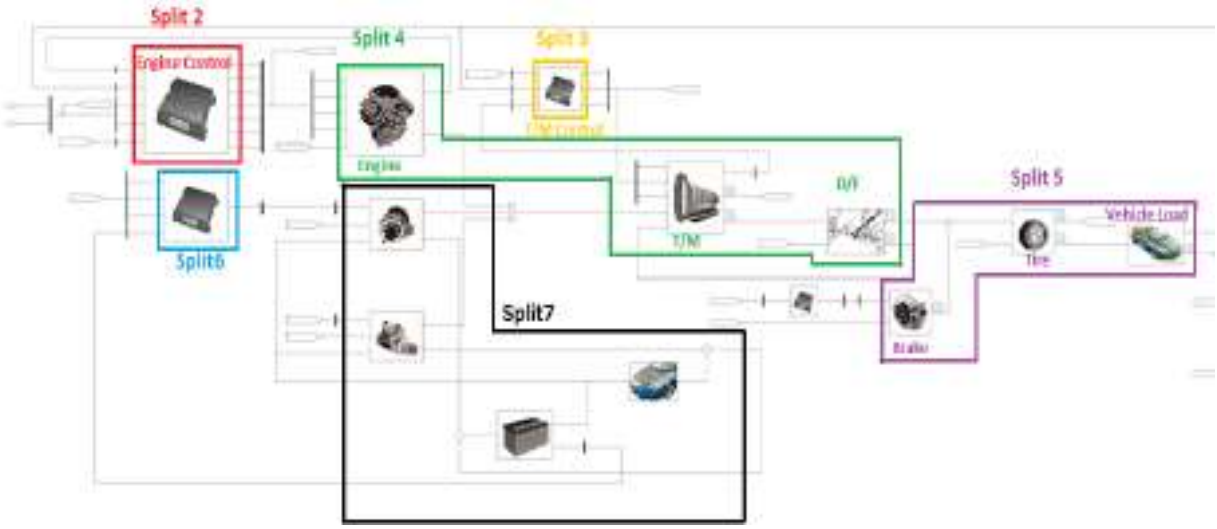


図 3.4.2　経産省ガイドライン準拠モデル Vehicle model の構成

まずは、使用するシミュレーションツールへ 7 つの FMU を取り込みます。取り込み方法は、メニュー画面で指定する場合や Explorer からドラッグ＆ドロップなどツールによって異なりますので、マニュア

ルを参照して下さい。

3.4.2. モデルの接続

表 3.4.1 を参考に各 FMU 間を接続します。黄色ハッチングは目標車速、緑色はモニタ用出力です。

図 3.4.3 はモデル接続例です。各 FMU の左側端子は入力、右側端子は出力になっています。モニタ用端子については接続に使用しません。使用するツールにより接続時の体裁は異なるためご注意ください。

表 3.4.1 各 FMU の入出力信号

FMU Number	出力端子			入力端子	FMU Number	FMU Name	出力端子	
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	2 n_eng_rpm	→	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	→	1 SOC_BT_PNT_Lo_PCT		6	ALT_CNT	target_volt_ALT_V	1
2	2 flag_fuelcut	→	2 flag_fuelcut					
2	3 flag_IdelStop	→	3 flag_IdleStop					
2	5 n_eng_rpm_sig	→	4 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

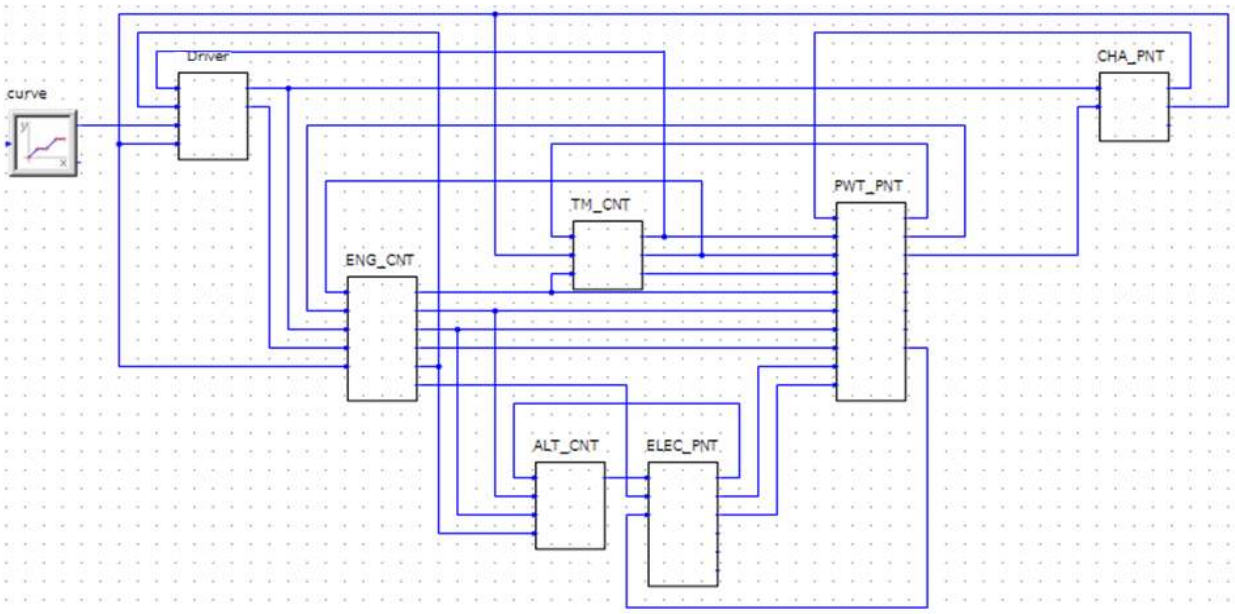


図 3.4.3　モデル接続例（SimulationX での例）

3.4.3. ドライバモデルの説明

ドライバモデル（FMU1:Driver）は、目標車速（表 3.4.1 黄色ハッチング部）となるようアクセルとブレーキを Closed Loop で操作します。今回は、JC08 モードで走行させるため jc08.csv の時系列データを入力します（図 3.4.4）。図 3.4.3 の接続例では curve へ設定します。

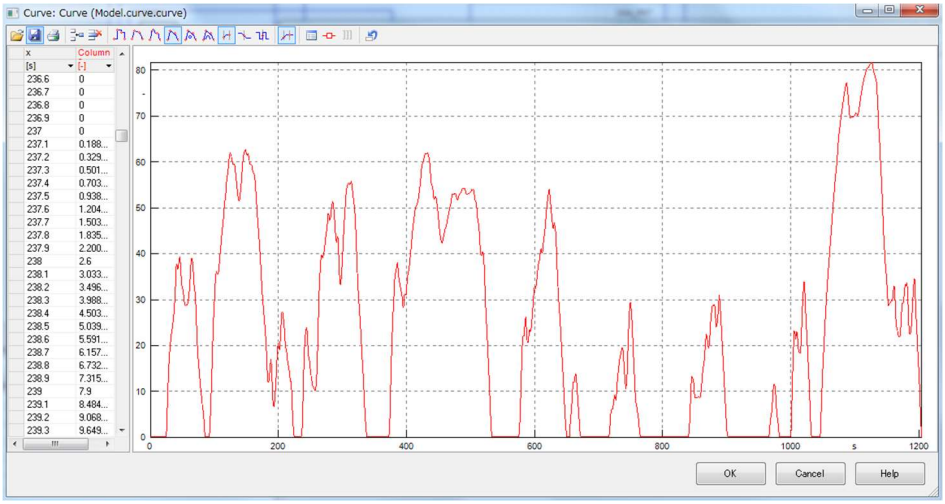


図 3.4.4　目標車速

3.4.4. マスタツールのシミュレーション設定

JC08 モードの所要時間である 1204 秒をシミュレーション時間として設定します。

ソルバは固定ステップの”ODE1”相当とします。

最少ステップサイズは、使用する各 FMU の最少計算間隔に合わせて”2.5e-3 [s]”とします。

Start Time	: 0 [s]
Stop Time	: 1204 [s]
Solver、 Step Sizes and Tolerances	: Fixed step Solver（固定ステップ）
Integration method	: Euler Forward （ODE1 相当）
Min. Calculation Step Size	: 0.0025 （または 2.5e-3）[s]
Min. Output Step Size	: 0.01 [s] ・・・出力結果のサンプル時間なので任意

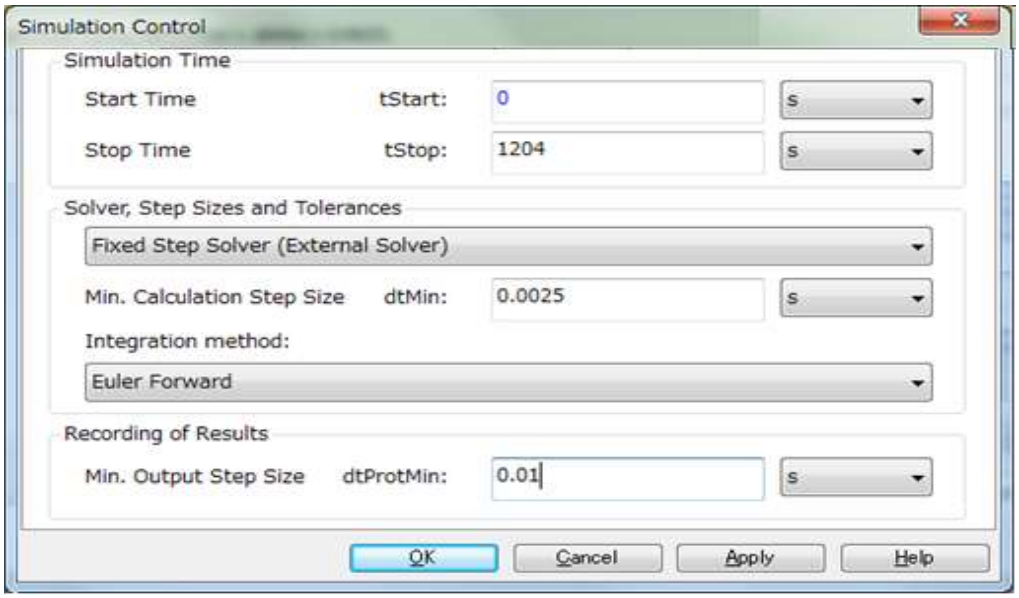


図 3.4.5　シミュレーション設定（SimulationX での例）

3.4.5. Communication Step Size の設定

Co-Simulation では FMU 毎に入力信号を受け取る時間間隔設定が必要となります。

今回のモデルでは全ての FMU を 0.0025（または 2.5e-3）[s]で設定します。

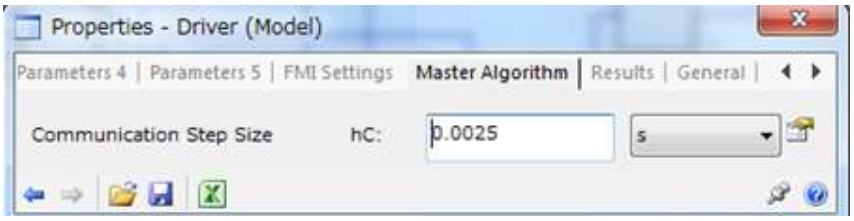


図 3.4.6　Communication Step Size の設定（SimulationX での例）

3.4.6. シミュレーション結果

FMU からの出力信号をいくつか掲載しました。

詳細は Sample_6p3.txt にシミュレーション結果を保存しているため参考にして下さい。

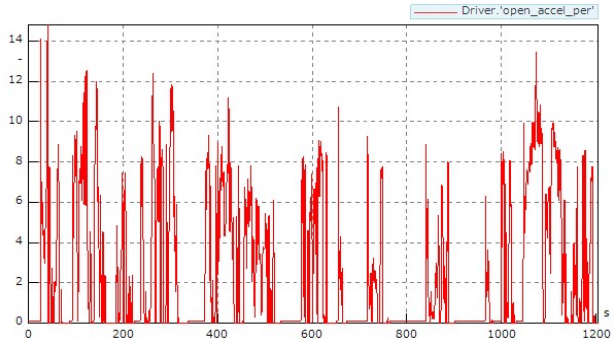


図 3.4.7 a　FMU1　open_accel_per

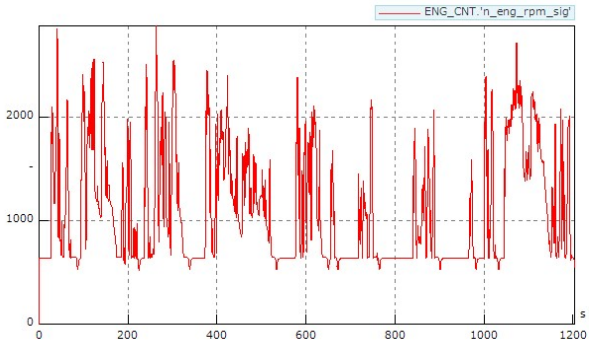


図 3.4.7b　FMU2　n_eng_rpm_sig

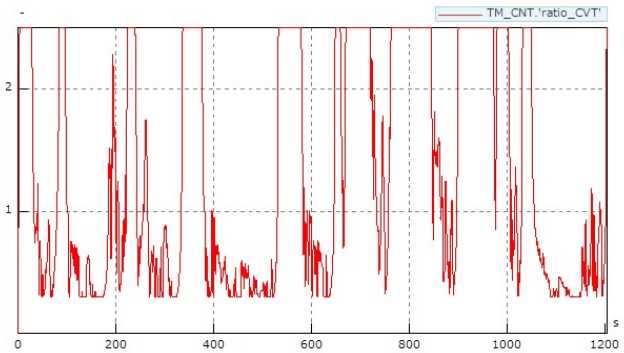


図 3.4.7c　FMU3　ratio_CVT

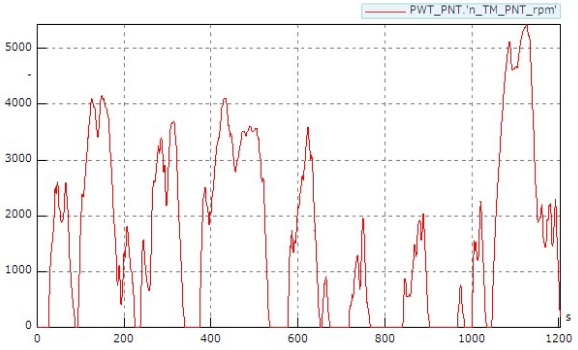


図 3.4.7d　FMU4　n_TM_PNT_rpm

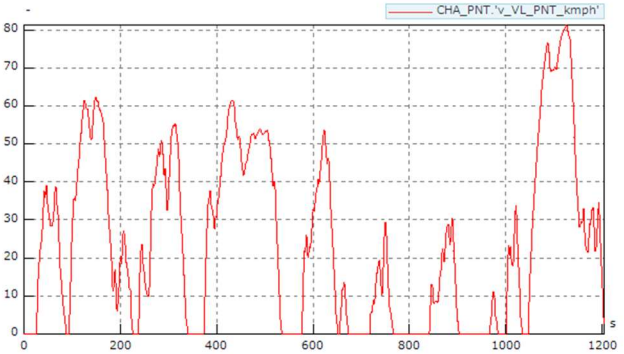


図 3.4.7e　FMU5　v_VL_PNT_kmph

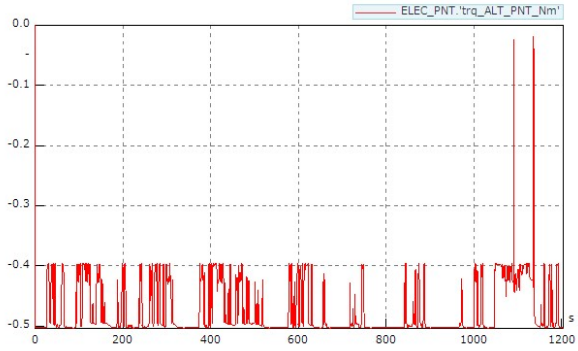


図 3.4.7f　FMU7　trq_ALT_PNT_Nm

参考文献

[1] 緒方洋介、村上晋太郎、他:FMI, Model Exchange, Co-simulation におけるシミュレーション安定性に関する考察、自動車技術会 2018 年春季大会 S4-4,(2018)

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

[3] 市原純一、斉藤春樹、他:複数モデリングツールによる FMI を用いた Co-Simulation に関するモデル接続活動紹介（第 2 報）、自動車技術会 2018 年春季大会 S4-2,(2018)