非因果モデリングツールを用いた FMI モデル接続ガイドライン

Ver.1.0

平成 27 年 3 月 19 日

変更履歴

Revision	日付	内容
1.0.0	2015/03/19	ガイドライン Ver1.0 リリース

目次

第1章　序章

1.1　はじめに

高機能，高信頼性が求められる自動車システムの開発においてシミュレーション活用の重要性が増してきている中，異なる物理領域や異なるシステム階層で使用されるシミュレーションモデルを自動車システム開発に関係する関連企業間で有効活用できる開発環境のニーズが高まってきている．シミュレーションモデルの記述には，様々なソフトウエアツールが使用されているが，上記の様に企業間でシミュレーションモデルを有効活用するためには，相互のソフトウエアツールを統一するか，ソフトウエアツール間でのモデル変換や接続インターフェース対応が必要となりシミュレーション実行を簡単に行えないという問題がある [1]．また，物理モデリングの方法として，回路図のような物理的事象のつながりを，物理要素間（例えば電圧源端子と抵抗端子）の接続としてモデリングできる非因果的モデリング（Acausal modeling）の活用も注目されるようになってきた [2].

このような背景を踏まえ，2012 年 3 月より自動車技術会共同研究センターに国際標準記述によるモデル開発・流通検討委員会が新設された．本ガイドラインは，同委員会のモデル接続技術検討 WG の活動を通してまとめられたものである.

本ガイドラインでは，非因果的モデリングツールを用いてモデルの分割と接続を行う場合を主体に扱っている．モデルの分割と接続に必要となる基本的な技術を述べた後，モデル接続に関する標準仕様となることが期待されている Functional Mock-up Interface（FMI）の概要と活用方法及び注意点について述べる．構成要素モデルを活用してシステムモデルを構築する際に，モデル分割ができない，あるいはモデル接続ができないなどの問題解決の参考になるガイドラインを目指して作成を行った.

FMI 仕様は基本的には FMI2.0 をベースとしている．モデルの例題は後述する代数ループが生じやすい電気回路を主体に説明しているが，基本的な考え方は他の物理領域へも置き換えが可能と考える．なお，他の物理領域での例題検討は今後の課題としたい.

本ガイドラインに記述されている Co-Simulation は FMI 仕様で定義されている FMI の種類の 1 つであり，異なるツールを相互接続したシミュレーションは連成シミュレーションと記述している．Co-Simulation の定義については第 5 章で説明する.

今後の自動車業界での企業間を越えたモデル活用及びモデル流通の加速に役立てて頂ければ幸いである.

本ガイドライン作成に協力頂いた企業，団体は以下の通りである.

AZAPA 株式会社，アンシス・ジャパン株式会社，ダッソー・システムズ株式会社，株式会社デンソー，同志社大学，トヨタ自動車株式会社，ニュートンワークス株式会社，株式会社本田技術研究所，マツダ株式会社（50 音順）.

なお，本ガイドラインは，自動車開発の手法や品質を保証するものではない．また，本ガイドラインに記載されている事項は予告なしに変更または廃止される場合がある．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する.

1.2　背景

自動車システムの一例として，ハイブリット車の構成概要図を図 1-1 に示す．この図にあるように，ハイブリット車はバッテリ，インバータなどの電気部品やモータ発電機などのメカトロ部品，そしてエンジンやトランスミッションなどの機械部品というように異なる物理領域の部品で構成されていることが分かる．また，各部品では電子制御が採用されることも多く，個別の制御ユニットを部品の構成要素に含めて，サブシステムを構成する場合もある．制御ユニットには，制御ソフトウエアが組み込まれ，ネットワーク接続を介して様々なデータのやり取りが行われる．

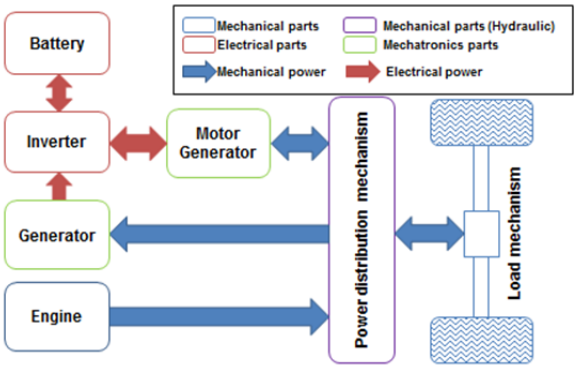


図 1-1　ハイブリット車の構成概要図

従来，このような自動車システムの開発はシステムの構成要素となる個別部品に対する要求仕様に基づいた試作を行い，それらを組み合わせた試作車を用いてテスト確認が行われ，テスト結果に応じて要求仕様の最適化を行うという繰り返しで進められてきた．しかし，機能の複雑化と開発期間の短縮という，相反する要望に対応していくためには従来型の開発スタイルでは限界が見えてきているのも事実である．そこで最近拡大してきているものが図 1-2 に示すようなシミュレーションテストの活用である．従来の仕様書と実部品という関係から，仕様書とそれに対応する実行可能モデルへの置き換えを行い，システムの構成要素となる実行可能モデルを組み合わせたサブシステムモデルや，さらに複数のサブシステムモデルを組み合わせた試作車モデルを用いてシミュレーションでのテスト確認とそれに応じた要求仕様の最適化の繰り返しを行える開発スタイルである．

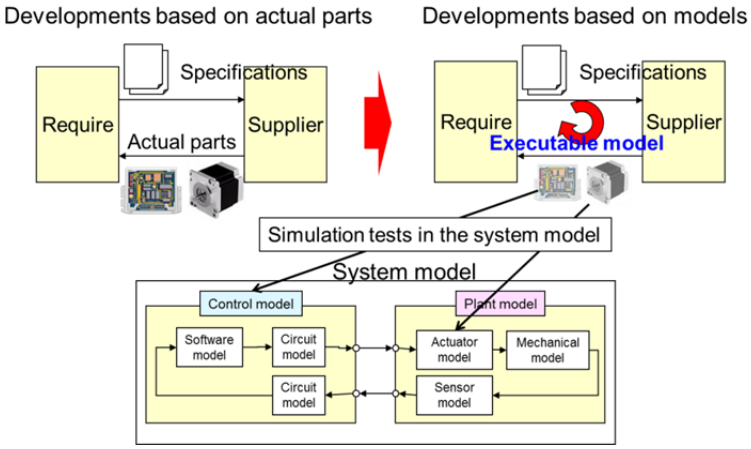


図 1-2 実行可能モデルを活用したシミュレーションテスト

そのためには，異なる物理領域や異なるシステム階層で使用されるシミュレーションモデルを自動車システム開発に関係する関連企業間で有効活用できる開発環境が必要である．シミュレーションモデルの記述には，様々なソフトウエアツールが使用されているが，企業間でシミュレーションモデルを有効活用するためには，相互のソフトウエアツールを統一するか，ソフトウエアツール間でのモデル変換や接続インターフェース対応が必要となる．

しかし，企業間でシミュレーションモデルを有効活用するためにツールを統一することは困難である．そこで異なるツールのシミュレーションモデルでも相互接続ができる技術がシミュレーションテストの活用拡大には必要不可欠となる．

1.3 ガイドラインの必要性について

1.1 節でも述べたように，モデルの作成と接続のためのインターフェース仕様に FMI を用いても，そのモデルは接続できない場合や接続できてもシミュレーション実行できない場合も考えられる．単一のツール環境であれば，モデル接続の不整合チェックをツールに委ねることも可能かもしれないが，異なるツール環境で作成されたモデルを組み合わせてサブシステムモデルや試作車モデルを構築する場合には，モデルを組み立てるエンジニアとしてのノウハウが要求される．そこでこのような問題を事前に回避するため，モデル接続に関係する基礎技術の理解とモデルの作成や接続にかかわるノウハウの把握ができるガイドラインが必要となる．

1.3.1 ガイドラインの目的

本ガイドラインの目的は，より高機能・高性能な自動車システムをタイムリに開発するために異なる部署や異なる企業で作成されたシミュレーションモデルを相互に接続し，システム動作シミュレーションを行う上で，必要となるモデル接続上の基礎的な技術要素や

注意点等を解説し，FMI の活用を促進することである.

1.3.2　ガイドラインの対象ユーザ

本ガイドラインでは，自動車システム開発においてモデルを活用したシステム動作予測を行おうとしているエンジニアを対象としている．カーメーカやシステムサプライヤ及び部品サプライヤで制御コントローラやプラントモデルの開発にかかわっているエンジニアを想定している．ツールの対象は，形状を前提としないモデル記述で数値シミュレーションが可能なものである．しかし，モデル接続の技術という観点からは３次元構造のモデルを用いた機構解析ツール等の他のツールにも応用が可能と思われる.

本ガイドラインでは，コントローラやプラントモデルなどの個別要素モデルの具体的なモデリング手法は対象外としている.

第2章　現状のモデル接続方法の概要

2.1　シミュレーション環境の構成要素

モデルを用いたシミュレーション環境を構成する要素には以下が含まれる．その構成を図示したものが図 2-1 である．

図 2-1 シミュレーション環境の構成要素

- モデル

グラフィカルなモデル記述やモデリング言語，プログラミング言語で記述された物理動作やコントローラ動作を記述したもの．数式記述，表，マップなども含む

- ソルバ

積分等の数値演算を行うための機能

- アプリケーションソフト

シミュレーションツールアプリケーション動作

- コンピュータ

アプリケーションソフトを実行するもので，ネットワーク通信環境も含む

2.2　シミュレーション環境の構成要素に対応したモデル接続形態

モデル接続を行う時の接続形態は，2.1 項のシミュレーション環境の構成要素のどのレベルで接続を行うかにより，以下の 4 つのパターンに分類することができる．それを図示したものが図 2-2 である．

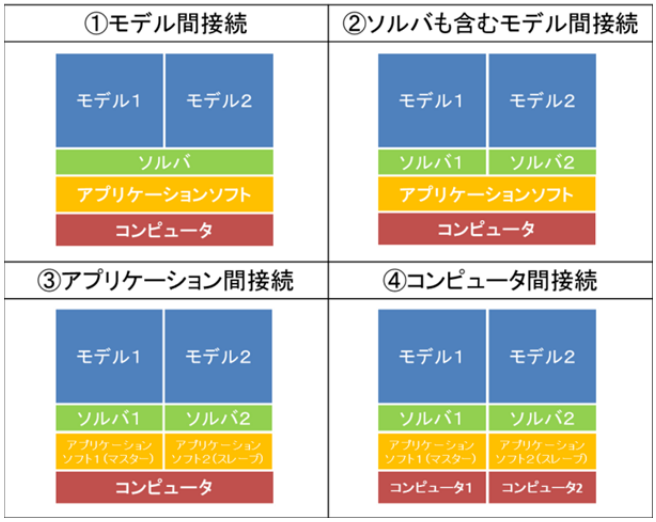


図 2-2 シミュレーション環境の構成要素に応じたモデル接続形態

① モデル間接続

同一のシミュレーションツールアプリケーション上で実行可能なモデル間の接続．個々のモデルにはソルバは含まれず，シミュレーションツールアプリケーションとして用意されているソルバが使用される．FMI の Model Exchange は，このモデル間接続に分類できる．FMI については第 5 章で説明する．

② ソルバも含むモデル間接続

同一のシミュレーションツールアプリケーション上で実行可能なモデル接続ではあるが，モデル自体にソルバが内蔵されているため，アプリケーションソフト側からは入力の値を渡し，ソルバ内蔵のモデルの計算結果を受け取ると言った動作となる．FMI の Co-Simulation は，このソルバも含むモデル間接続に分類できる．FMI については第 5 章で説明する．

③ アプリケーションソフト間接続

2 つ以上のシミュレーションツールアプリケーション間で相互にシミュレーション結果のデータをやり取りする接続．接続のためにはシミュレーション実行の時間管理等を行うマスタのツールとそれに従うスレーブのツールが必要となる．マスタのツールは必ずしもモデルやソルバを必要としない．また，マスタ，スレーブ共にソルバを内蔵したモデルを接続することも可能である．一般的に連成シミュレーションと呼ばれているものはこのアプリケーションソフト間接続に分類できる．

④ コンピュータ間接続

複数 CPU コアやネットワークを介した複数のコンピュータを接続してシミュレーション演算を並列処理可能な接続形態．マスタとスレーブのツールが必要となる点は③のアプリケーションソフト間接続と同じである．

2.3　シミュレーションモデルの構成要素として使用できるファイルの分類

目的とする計算結果を得るために，シミュレーションの構成要素を接続して，サブシステムモデルやそれを組み合わせた車両システムのような上位階層レベルのシステムモデルを組立てるためには，構成要素をファイルとして保存して，下位階層のモデル設計者から上位階層レベルのサブシステムや最上位システムのモデル設計者に渡す必要がある．その際のファイルとして使用可能なものを大まかに分類したものが表 2-1 である．これらのファイルは個別のシミュレーションツールでそのまま読み込めるものもあるが，取り込むための個別インターフェースが必要なものもある．

表 2-1 シミュレーションモデルの構成要素として使用できるファイルの分類

No.	ファイル分類	例
1	個別ツールに依存するファイル	● ツールごとに設定された拡張子のモデルファイル
2	モデリング言語ファイル	● VHDL-AMS で記述されたモデルファイル ● Modelica で記述されたモデルファイル（グラフィカル表現も含む）
3	プログラミング言語ファイル（モデリング言語依存のないファイル）	● C ファイル ● C++ファイル ● Fortran ファイル
4	コンピュータの OS 上で実行可能なライブラリファイル	● dll ファイル（Windows） ● so ファイル(Linux)

2.3.1　個別ツールに依存するファイル

このファイル形式分類は，個別ツールに依存したファイル形式である．したがって，この形式で記述されたモデルを用いてモデル接続が行えるのは，同一ツール環境のみとなる．もしモデルを共有する相互間でツール環境を統一できるのであれば，モデルファイルがそのまま読み込めるのでモデル接続が一番行いやすいファイル形式である．

2.3.2　モデリング言語ファイル

このファイル形式分類は，ハードウェア記述言語である VHDL-AMS で記述されたモデルファイルや物理モデリング言語である Modelica で記述されたモデルファイルなどである．これらのモデリング言語ファイルを用いてモデル接続が行えるのは，同一のモデリング言語が扱えるツール間となる．ただし，モデル記述に使用される要素部品のライブラリのバージョンが異なると，モデル接続がうまく行えない場合があるため，モデル交換を行う相

互間でモデル記述に使用するライブラリを揃えておくことが必須条件となる．また，モデル内のパラメータデータとしてデフォルト値を使用している場合，ツールごとにデフォルト値の設定が異なる場合があるため，パラメータデータは明示的に設定を行うことが望ましい．モデルを共有する相互のツール環境が同じモデリング言語ツールであるならば，モデリング言語レベルのファイルでモデル接続を行うことが可能である．モデリング言語の概要については第 4 章で説明する．

2.3.3　プログラミング言語ファイル

このファイル形式分類は，C・C++・Fortran などの一般的なプログラミング言語で記述されたファイルである．これらのファイルを用いてモデル接続を行うためには，ツールごとに接続インターフェースを用意する必要がある．モデルを実行するコンピュータ環境が異なるような場合にも，これらのファイル形式を用いてモデル接続を行うことが可能である．なお，コンパイラーのオプション指定の違いには注意が必要である．

2.3.4　コンピュータの OS 上で共通利用可能なプログラムファイル

このファイル形式分類は，Windows 上で実行できる dll ファイルのような OS 上で共通利用可能なプログラム部品ファイルである．これらのファイルを用いてモデル接続を行うためには，ツールごとに接続インターフェースを用意する必要がある．また，下位階層のモデルの設計者は，シミュレーションツールでモデリングしたモデルからファイルを作成するために，コンパイル作業を行う必要がある．FMI ではこの接続インターフェースの仕様を規定している．FMI の Model Exchange や Co-Simulation モデルの実体は，Windows OS の場合には dll ファイルとなる．（FMI については第 5 章を参照）．

第3章　モデル接続に必要となる基本技術

本章では，モデル接続における問題を理解するための前提となる，モデルのシミュレーション技術に関する概要を説明する．

3.1　常微分方程式（ODE: Ordinary Differential Equation）と微分代数方程式（DAE: Differential Algebraic Equation）

一般に，物理モデルで記述される式は，微分方程式と，微分演算を含まない代数方程式から成るが，そのモデル記述法としては，大きく，常微分方程式 (ODE)表現と微分代数方程式 (DAE)表現がある．常微分方程式表現とは，一般に，(3-1)式の形で表される状態変数 x，補助変数 u，時間 t についての常微分方程式と，微分を含まない変数間の拘束条件を表す代数方程式(3-2)式から成る．なお，モデル接続で扱うモデルのように，補助変数 u が外部から陽に与えられる入力変数となる場合，(3-2)式の代数方程式は現れない場合もある．

$$dx/dt = f(x,u,t) \tag{3-1}$$

$$0 = g(x,u,t) \tag{3-2}$$

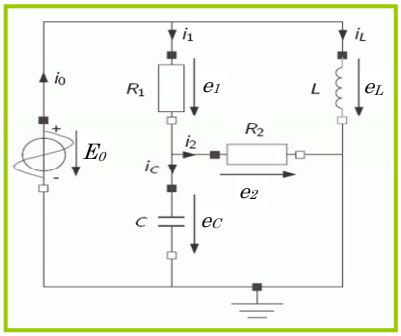
(3-1)式では，x の微係数 dx/dt が陽に定義されているため，そのままの形で ODE 用のソルバで数値積分を行って解くことができる．

一方，微分代数方程式表現とは，(3-3)式の様に，微係数 dx/dt も含んだ拘束条件式で表現されるものである．

$$0 = F(dx/dt, x, u, t) \tag{3-3}$$

(3-3)式を解くには，(3-1)式の形に解析的に式を変形してから ODE ソルバで解く方法と，(3-3)式の形のまま，陰解法を用いた DAE ソルバで解く方法の二つがある．種々のシミュレーションツールでは，いずれかの方法が実装されている．なお，それぞれのソルバで使われる積分アルゴリズムの概要については，後述する．

物理モデリングの方法としては，大きく，因果的モデリング(Causal modeling)と，非因果的モデリング(Acausal modeling)の２種類が存在する．因果的モデリングとは，入力と出力が何かを予め指定するモデル作成手法である．作成方法としては，物理モデル方程式をユーザが ODE 形式で手で解いた後，Simulink®などのように，それらの式をブロック線図形式で記述していく方法が一般的であるが，各コンポーネント間でやり取りする物理量とその入出力関係をあらかじめ指定する Bond Graph のような手法もあり，DAE を扱うこともできる．一方，近年，各コンポーネントの間でやり取りされる物理量を定義しておき，各コンポーネントの中では，それらの物理量の間で成立する関係式のみを定義し，それらの部品をあたかも実物のように組み合わせることで，全体システムのモデルを作成する手法が提案されてきた．このようなモデリングの方法は，コンポーネントへの入力，出力信号をあらかじめ決定しなくてよいため，非因果的モデリングと呼ばれる．非因果的モデリングでは，モデル記述形式としては，一般に DAE となる．



コンポーネント定義式:

$E_0 = f(t)$, $i_C = C \cdot du_C/dt$

$e_1 = R_1 \cdot i_1$, $e_L = L \cdot di_L/dt$

$e_2 = R_2 \cdot i_2$

接点方程式:

$i_0 = i_1 + i_L$, $i_1 = i_2 + i_C$

閉路方程式:

$E_0 = e_1 + e_C$, $e_L = e_1 + e_2$

$e_C = e_2$

図 3-1　非因果的モデリングによるモデル記述の例

ここでは，非因果的モデリングによるモデルの記述法について，概要を説明する．例として，図 3-1 の電子回路のモデルを考える．（電圧，電流の向きは電気工学の慣例とは逆だが，引用文献の記述を踏襲するため，容赦されたい．）電気系のモデルにおいては，端子間でやり取りする物理量として電圧 e と電流 i が定義されている．そして，例えば，抵抗素子内では，電圧と電流の関係式である $e = R \cdot i$ (R は抵抗）が定義されている．そして，回路図と同様に，グラフィカル・エディタ上で部品を組み合わせて回路モデルを作成すると，回路全体をシミュレーションするための方程式群が自動的に生成される．この際，接続した部品同士の間では，キルヒホッフの電流則（任意の節点で接続された電流の総和が零；接点方程式）とキルヒホッフの電圧則（任意の閉回路上の電圧の総和が零；閉路方程式）に相当する定義式が追加される．図 3-1 の例では，図の右半分にあるように，１０個の関係式が生成される．ここで，これらの式は，変数間の関係のみを表し，代入式ではないことに注意してほしい．非因果的モデリングでは，このように，代入式の右辺に表れる計算入力と，左辺に表れる計算出力を陽に意識する必要はない．実際の計算処理に用いられる数値解法としては，１ステップごとに ODE ソルバと Newton-Laphson 法などの反復計算による代数方程式ソルバを交互に解く DAE の解法が用いられる．また，数式処理による INDEX 低減アルゴリズムを用いて ODE に変換してから ODE ソルバで解く方法もある．

3.2　積分アルゴリズムについて

3.2.1　陰解法と陽解法

微分方程式を計算機で解くには，微分を差分で近似して，差分方程式を数値解法で解く方法が用いられる．この際，前進差分を用いる陽解法（Explicit method)と，後退差分を用いる陰解法（Implicit method)がある．一般に，陽解法では，系の時定数に対して，時間刻みを十分小さくしないと，解が不安定となり，振動や発散を起こすことと，また逆に小さくしすぎると誤差が累積することが知られている．一方，陰解法では，原理的に，解はよ

り安定となり，時間刻みがより大きくとれる．ただし，陰解法は，解について反復計算による収束計算が必要となり，また，収束許容誤差の設定が精度に影響を与えることになる．HILS(Hardware In the Loop Simulation)の様に，実時間計算が求められる場合，計算時間の制約上，陽解法しか使えない場合があるが，時間刻みの設定には十分注意が必要となるし，そもそも，系の時定数が小さい場合，実時間計算が不可能な場合もある．また，後述するスティフな系の場合，時間刻みを可変とした積分アルゴリズムを用いた方が実用的である．対象とする系の時定数に応じた，適切な積分アルゴリズムと，時間刻みや収束許容誤差の設定が，シミュレーション結果の精度に大きな影響を与えるので，注意が必要である．

表 3-1 に，主な数値積分法の分類を示す．また，代表的な数値積分アルゴリズムの特徴を表 3-2 に示す．

表 3-1　主な数値積分法の分類

	陽解法（Explicit）	陰解法（Implicit）
一段階法 (Single Step)	前進 Euler 前進 Runge-Kutta	後退 Runge-Kutta
多段階法 (Multi Step)	Adams-Bashforth	Adams-Moulton 後退微分公式(BDF)

表 3-2　主な数値積分アルゴリズムの特徴 [3]

アルゴリズム	方程式	スティフ系対応	時間刻み	手法
Euler	ODE	No	固定	前進 Euler
RKF45	ODE	No	可変	前進 Runge-Kutta
DOPRI5	ODE	No	可変	前進 Runge-Kutta (Dormand-Prince)
DIRK	ODE	Yes	可変	後退 Runge-Kutta
SIRK	ODE	Yes	可変	後退 Runge-Kutta
Radau IIa	ODE	Yes	可変	後退 Runge-Kutta
LSODAR	ODE	Yes	可変	多段階 Adams 法
DASSL	DAE	Yes	可変	BDF
RADAU5	DAE	Yes	可変	BDF

3.3　スティフ系における問題

モデル全体に含まれるダイナミクス（動特性）の時定数のオーダーが大きく異なっている場合，スティフ系と言われる．例えば，機械系と油圧系の物理モデルをそのまま結合した場合など，それぞれの系の時定数が大きく離れているため，スティフ系になり易い．固

定刻みの陽解法の数値積分法（前進 Euler 法，前進 Runge-Kutta 法など）を用いる場合，最も小さい動特性の時定数に合せて時間刻みを設定しないと，計算が不安定となったり，結果が振動したりする．しかし，この方法では積分ステップ数が膨大になる．スティフ系のシミュレーションを行う場合は，スティフ系に対応した陰解法数値積分アルゴリズムを用いる必要がある．（
表 3-2 参照.）

3.3.1 可変時間刻みアルゴリズム

ここではほとんどの VHDL-AMS 系システムシミュレータが採用している 2 種類の可変時間刻みアルゴリズムを説明する.

（１） 反復回数アルゴリズム

最も単純な時間刻みアルゴリズムであり，許容誤差内に収めるために必要になる反復回数を基にして時間刻みを計算する．このアルゴリズムは最小反復回数パラメータ以下で収束していれば時間刻みを増加させ，最大反復回数パラメータを超えるとタイプステップを減少させる.

（２） Local Truncation Error (LTE) 局所的打切り誤差アルゴリズム

Local Truncation Error 時間刻み方式では，テーラー級数近似を用いて，過渡解析の次の時間刻みを計算する．この方式では，局所的切り捨て誤差の許容値を使用して，内部時間刻みを予測する．予測された時間刻みでは許容誤差を満足しない場合，シミュレータは一つ前の時間刻みに戻り，時間刻みを短くし再度計算を行う．予測された時間刻みが許容誤差を満足する場合，シミュレータは時間刻みを次の計算タイミングで大きくする．多くの回路シミュレータはこのアルゴリズムをデフォルトで使用している [4].

3.3.2 最大時間刻み時間のデフォルト値の違いによる計算精度

上記のように時間刻みによる計算精度及び計算量は相反関係にある．一般的に時間刻みを小さくすると計算精度は改善されるが計算量が多くなる．時間刻みを大きくすると計算量は少なくなるが計算精度は悪化する．したがって時間刻みを必要以上に小さくすることは避け，精度に影響が出ない範囲で大きくすることが望ましい．可変時間刻み法を採用すると最小時間刻み時間 tmin と最大時間刻み時間 tmax の範囲内で時間刻みが変化するが，tmax,tmin のデフォルト値はツール環境によって異なるため，モデル交換の際には注意が必要である．例えば，tmax は Modelica 系のあるツールの場合，（解析終了時間－解析開始時間）/100 であるが，VHDL-AMS 系のツールでは入力波形周期の 1/10 であったり，1 /50 だったり様々である．周期の 1/10 という tmax の設定は大きすぎてサイン波のような解析では計算精度に影響が出ることがある（SPICE 系シミュレータは多くは周期の 1/50）.

3.4 代数ループについて

代数ループ（Algebraic loop）は，代数方程式によって表現されるループのことで，具体

的には，直接フィードスルーをもつブロックにおいて自分自身のブロック出力をフィードバックで入力するような場合に生じる．つまり，入力と出力が同タイミングで要求されるような状態である．"代数" は微分係数を含まない方程式を意味する． [5] [6]

代数ループの簡単な例として，ｙ＝ｘ－ｙの関係を表したブロック線図例を図 3-2 に示す．

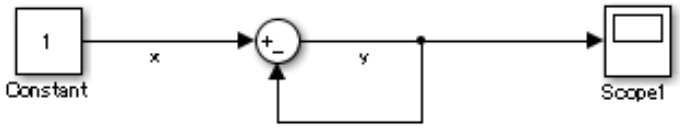


図 3-2　ｙ＝ｘ－ｙの関係を表した代数ループのブロック線図

このモデルの正解は，ｙ＝ｘ－ｙを陽に解いたｙ＝ｘ／２（＝０．５）となるが，この操作を自動で行うには，数式処理の機能が必要である．代数ループを含む系を数値的に解くには，ｙの推定値を，ある初期値（例えば，*y[0]=0*）から初めて，ｙの定義式から計算される値 *y_[0]=x−y[0]=1−0=1* と，今回の推定値 *y[0]=0* の差 *Δｙ=y_[0] − y[0] = 1−0 = 1* が零に近づくように，ｙの推定値を修正する収束計算式

y[k+1] = y[k] + r(y_[k]-y[k])*　　（ここで，*r*（*0<r<1*）は，収束係数）

を用いた代数ループソルバを用いる必要がある．この場合，反復計算のための時間がかかることになり，代数ループソルバの性能により，解の精度も影響される．したがって，これらのツールでは，代数ループにならないように，あらかじめ数式処理により陽に解いたモデルを作成することが望まれる．なお，Modelica 系ツールの一部では，線形の簡単な代数ループは，数式処理によりあらかじめ解いてから計算するものもある．

しかし，一般には，上記の例のように，簡単な数式処理で解けないモデルも多くあり，一方，反復計算量を削減したいというニーズも存在する．このため，近似的に，代数ループを回避するために出力値を遅延させ，図 3-2 の例に対して図 3-3 のように，$y_{n+1}=x_{n+1}-y_n$として計算させる対処法も提案されている．しかし，この場合，初期値が真値と異なると，正確な解に収束しない場合が存在する．図 3-3 のモデルの計算結果を図 3-4 に示す．計算結果が振動し，期待する結果と異なることが分かる．

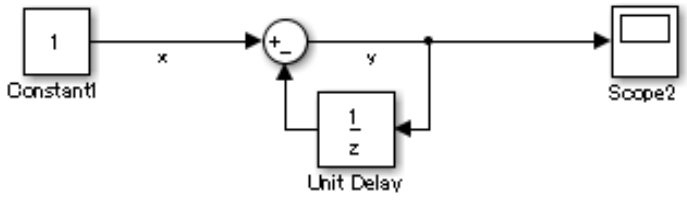


図 3-3　代数ループ除去に出力の遅延を適用したブロック線図

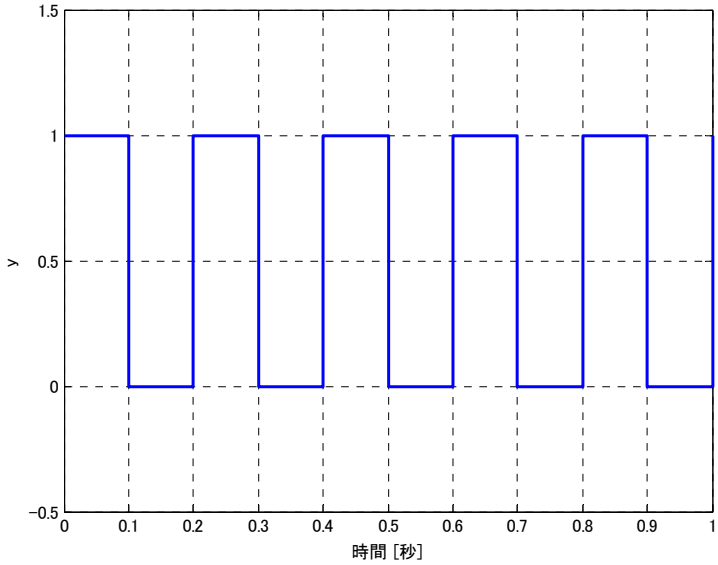


図 3-4　出力値の遅延(0.1 秒)を適用したブロック線図の計算結果

一般に，代数ループ内を一巡した時のゲイン A の絶対値が，$|A| < 1$ ならば真値に収束するが，$|A| = 1$ ならば持続振動，$|A| > 1$ ならば発散する．（図 3-3 の場合は，一巡伝達関数ゲインが 1 のため，持続振動した．）HILS への適用など，計算時間の制約から近似的に遅延を入れたモデルを作成する場合，この点に注意が必要である．

また，様々なツールを用いて作成された異なるモデルを接続して一つの大きなモデルを作る場合も，全体モデルの中に代数ループができないようなモデル構造とすることが重要となる．どうしても代数ループを回避できず，上記の遅延を入れた対処法を取る場合，ループの一巡伝達関数ゲインが，上記の収束条件を満たすか，確認することが必要である．ただし，ループ内に非線形要素が含まれる場合，一巡伝達関数ゲインを一意に決定することができない場合もあるため，代数ループソルバの適用範囲には注意が必要となる．

3.5　スルー変数とアクロス変数について

物理モデリングにおいて，モデル間の接続を考える際に重要となるのが「スルー変数（通過変数）」と「アクロス変数（横断変数）」である．スルー変数とは，「キルヒホッフの第一法則（電流則）」を満たす変数，すなわちモデル上の任意の節点に流れ込む量の総和が 0 となる変数であり，機械系（並進系）における「力」や電気系における「電流」がこれに相当する．一方アクロス変数とは，「キルヒホッフの第二法則（電圧則）」を満たす変数，すなわちモデル上の任意の閉路に沿って積算した総和が 0 となる変数であり，機械系（並進系）における「変位」あるいは「速度」や電気系における「電圧」がこれに相当する（図 3-5）．物理モデリングにおいてはこれらを対として扱い，各モデルの入出力変数として使用されることが一般的である（表 3-3）．

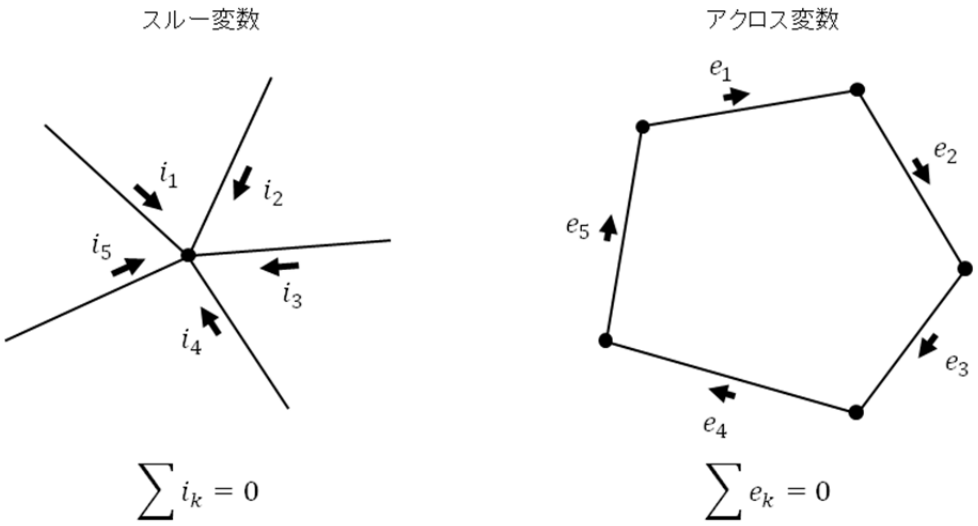


図 3-5 スルー変数とアクロス変数の概念図（キルヒホッフの法則）

表 3-3 スルー変数とアクロス変数の例

	スルー変数	アクロス変数
並進機械	力[N]	速度[m/s]
回転機械	トルク[Nm]	角速度[rad/s]
電気	電流[A]	電圧[V]
流体	流量[m^3/s]	圧力[Pa]

3.6　因果的モデリングと非因果的モデリング

3.1 節で述べたように，物理モデリングの手法には因果的モデリングと非因果的モデリングがある．ここでは両者の特徴と，FMI が前提としている因果的接続において起こり得る問題（信号極性問題）について述べる．

例えば，理想的な電気抵抗をモデル化する場合，「電流を流すとそれに比例した電圧降下を起こすモデル」と「電圧降下（電位差）を与えるとそれに比例した電流が流れるモデル」の 2 通りの入出力の与え方が考えられる．このように「入力＝電流，出力＝電圧降下」あるいは「入力＝電圧降下，出力＝電流」と，モデル作成時にその入出力を定義するモデリング手法を「因果的モデリング」と呼ぶ．同様に，複数のモデルを接続する時に「一方のモデルが出力した電圧を他方のモデルが入力として受け取る」というように，個々のモデルの入出力を明確にして接続する手法を「因果的接続」と呼ぶ．対して，電気抵抗をモデル化する場合に，「電圧降下と流れる電流は比例関係にある $(E = IR)$」とだけ定義し，入出力変数を明確に定義しないモデリング手法を「非因果的モデリング」と呼び，複数のモデルを接続する時に「接続する端子間の電圧が等しい」というように，入出力を明確に定

義しない接続手法を「非因果的接続」と呼ぶ.

因果的モデリングにおけるモデル式は，モデルの入力変数を$\boldsymbol{u}$，出力変数を$\boldsymbol{y}$とすると，$\boldsymbol{y} = f(\boldsymbol{u})$と表される．この式中の「＝」は代入記号を表しており，右辺の計算結果を左辺に代入する操作を表している．一方，非因果的モデリングにおけるモデル式は$\boldsymbol{x}$を状態変数として，$f(\boldsymbol{x}) = 0$と表される．非因果的モデリングでは入出力変数は明確ではないため，$\mathbf{x}$の中には他のモデルとの接続に使用される変数も含まれる．非因果的モデリングの式中の「＝」は左右が等価であることを示す等号であり，代入処理を表すものではない.

3.6.1　因果的接続のメリット・デメリット

因果的接続（因果的モデリング）では上記の通り，各モデルの入出力が明確であるために積分ステップごとの計算順序を追跡し易いというメリットがある．また，モデル作成者が計算順序を（積分アルゴリズムに関与しない範囲で）制御することが可能である．ただし，上記の通り因果的モデリングでは代入処理によってモデル式を表現するため，以下のようなデメリットがある.

① 単純な方法では代数拘束条件を表現することができない.（すなわち DAE を表現することができない）
② モデルを接続するときは一方のモデルの入力変数と他方のモデルの出力変数が対応していなければならない,

①の結果として，因果的接続（因果的モデリング）においては通常，モデル作成者が DAE を ODE に変換した上でモデルを記述しなければならない（ツールによっては因果的接続を行う場合でも DAE を表現がすることができるできるものもある）. また, ②の結果として，同じモデルを直接接続することはできない．これらを電気抵抗のモデルで例えると，①は「電気抵抗同士の並列接続ができない」ことに相当し，②は 「電気抵抗同士の直列接続ができない」ことに相当する（図 3-6）．因果的接続では，これらを解消するために接続形態に応じて入出力変数を適切に変更しなければならない.

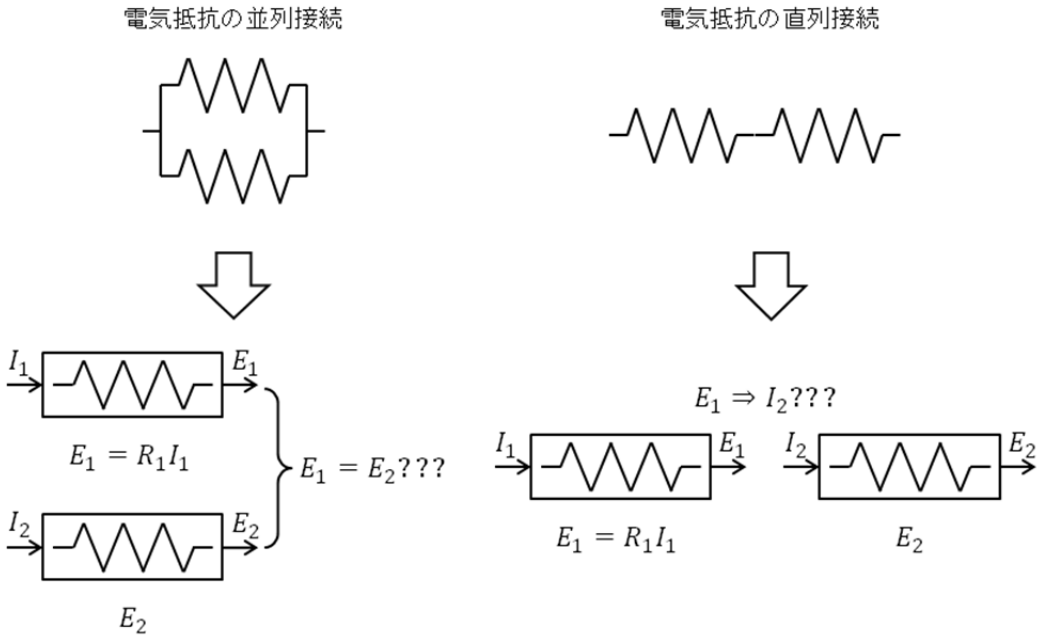


図 3-6 電気回路における因果的接続の問題

3.6.2 非因果的接続のメリット・デメリット

非因果的接続（非因果的モデリング）は因果的接続の裏返しである．すなわち代数拘束条件を表現することが可能であり，同じモデルを直接接続することも可能である（図 3-7）．ただし，あくまで接続が可能なだけであり，そのモデルを実行可能か，十分な計算精度が得られるかはモデルの記述方法とツールのソルバの性能に依存する．また，非因果的接続（非因果的モデリング）では，モデル作成者は各変数およびその微分値の関係式を方程式で表現するだけであり，計算順序を制御することができず，追跡も困難であることが多い．

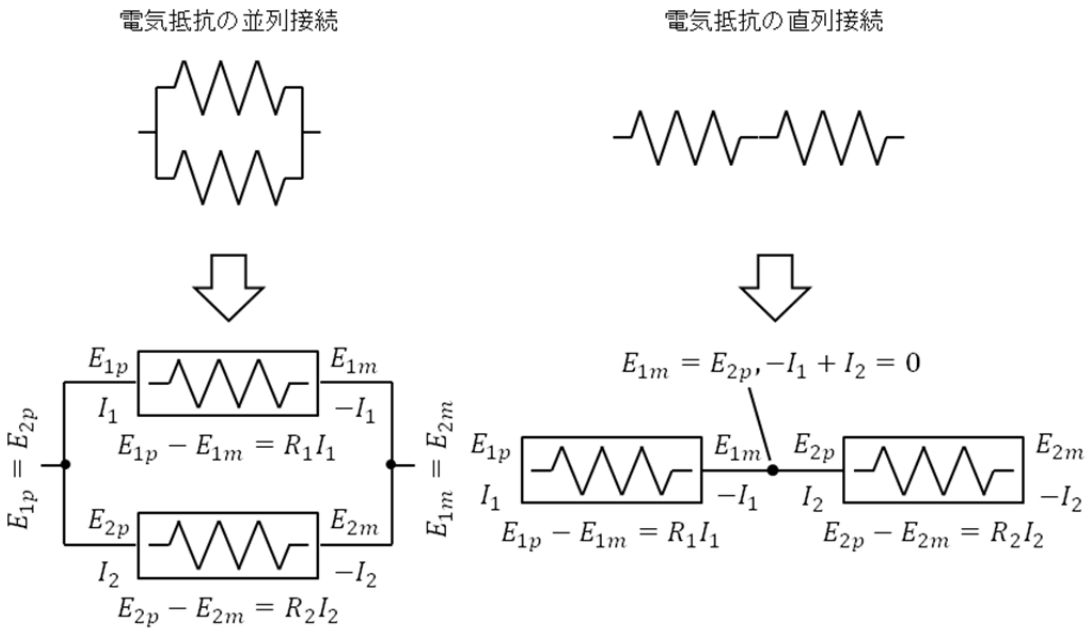


図 3-7 電気回路における非因果的接続

3.6.3 信号極性問題

例として, 図 3-8 のように内部抵抗をもった電圧源と電気抵抗を接続した回路をモデル化することを考える. 電源の正極の電流をI_b, 電位をE_bとする. 同様に抵抗の正極の電流をI_r, 電位をE_rとする. 電流の向きはそれぞれその要素に流れ込む向きを正とする.

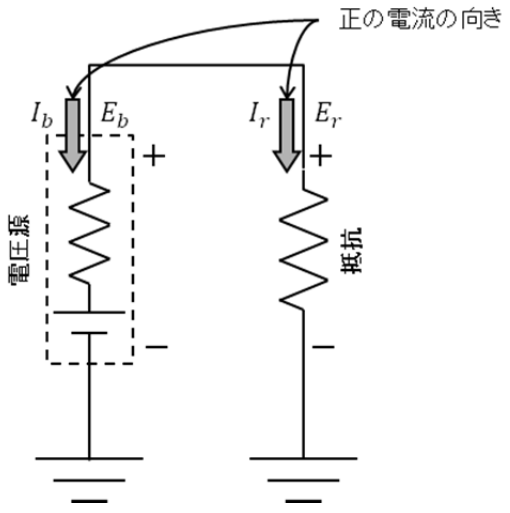


図 3-8 電圧源と抵抗による回路（回路図）

まず，非因果的接続を考える．非因果的接続では端子間を接続することは，その端子間でキルヒホッフの法則が成立することを意味するため，電源モデルと抵抗モデルの正極を接続すると，以下の連立代数方程式が立てられる（図 3-9）．

$$I_b + I_r = 0 \tag{3-4}$$

$$E_b = E_r \tag{3-5}$$

電流に関して，(3-4)式から，$\boldsymbol{I_r} = 1\mathrm{A}$の時，$\boldsymbol{I_b} = -1\mathrm{A}$となり，符号が反転している．このような反転はキルヒホッフの法則（第一法則）によるものであり，端子間を接続することによって（(3-4)式が立てられることによって）「極性の反転」が自動的に行われることが非因果的接続の特徴である．一方，電位に関しては(3-5)式から信号極性が反転せず，等式が成立していることがわかる．

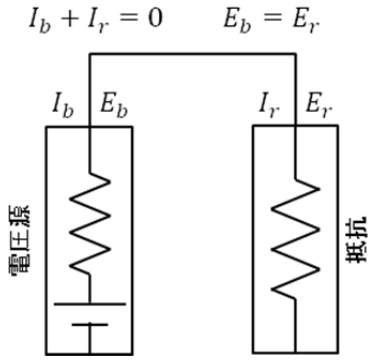


図 3-9 非因果的接続の例（モデル図）

次に因果的接続の場合を考える．端子間の接続は「代入式」を意味する．そのため，あるモデルの出力電流端子と他のモデルの入力電流端子を接続すると，それらの値が等しくなり，当然のことながら上記のような信号極性の反転は生じない（図 3-10）．それぞれのモデルの端子に流れる電流を「流れ込む向きを正」という統一した規則で極性を定義していた場合，本来は(3-4)式が成立すべきところが，因果的接続では$\boldsymbol{I_b} = \boldsymbol{I_r}$となってしまい，図 3-8 の回路を表現する正しいモデルとならない．これを回避するためには$\boldsymbol{I_b}$あるいは$\boldsymbol{I_r}$，のどちらか一方の符号（極性）を反転させて定義しておかなければならない．本ガイドラインではこのような問題を「信号極性問題」と呼ぶ．この問題は電流に限らず全てのスルー変数において生じるものである．

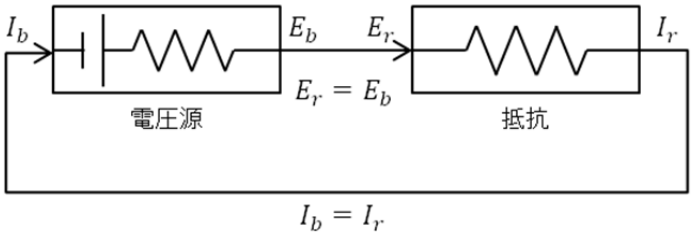


図 3-10 因果的接続の例（モデル図）

この例に限った最も簡単な回避方法を述べておく. この問題の原因は電圧源で定義したI_bの向きと，抵抗で定義したI_rの向きが不一致であることである（図 3-8）．これを回避するためには，あらかじめ回路の全体像を意識し，組み上げた時に電流の向きに不整合が生じないよう，極性を定義しておけばよい（図 3-11）．ただし，実際のモデル流通の場面においては必ずしも全体像が明らかではなく，また回路素子などのモデルの場合，一つの回路内で複数使用され，その全てが同じ向きに統一できるとは限らないため，有効な回避方法とは言えない. より一般的なケースにおいて不整合を生じない回避方法については 6.2.1 項で述べる.

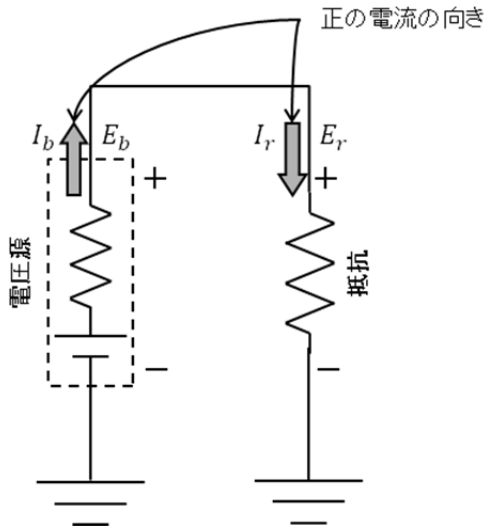


図 3-11 因果的モデルでの極性不一致の解消方法（回路図）

第4章　モデリング言語について

システムを計算機でシミュレーションするためには，対象となる系を定義するためモデリングを行い，プログラムのモデリング言語で記述し，それを実行するソルバに通知しなければならない．従来は C 言語や FORTRAN 言語等を用いて直接マトリクスの構築や求解法を記述したが，近年ではシミュレーションツールの進化により上位言語を用いてモデルを容易に定義することが可能になった．システムのシミュレーションモデルを記述するための言語として，VHDL-AMS と Modelica の背景と概要について記載する．

4.1　VHDL-AMS

4.1.1　背景

VHDL-AMS はシステム及び物理現象を表現し，シミュレーションに適した形式で記述できるように IEEE（Institute of Electrical and Electronic Engineers）にて標準化されたモデル記述言語であり，1981 年に米国防総省にて規定された VHSIC に端を発する．これは当時デジタル IC の増大による検証の不備が多発し，国防総省に納品されるデジタルシステムに対する統一されたモデル記述言語が求められたためである．これを受け，1993 年にデジタルプロセスの記述言語として VHDL (IEEE std. 1076)として規定された．

その後，アナログ IC の台頭によりシステム記述の重要性が再認識され，その複雑さと非線形性を表現するために新たなモデルの記述が求められた．特に欧州及び米国の自動車車載系を中心として要望が集められ，1999 年にデジタルハードウェア記述言語 VHDL をアナログに拡張した規約，VHDL-AMS (Very High Speed Integrated Circuit Hardware Description Language –Analog and Mixed Signal) (IEEE std. 1076.1-1999)が策定された．その後，国際標準規格(IEC 61691-6)として規定され，2009 版が最新である．

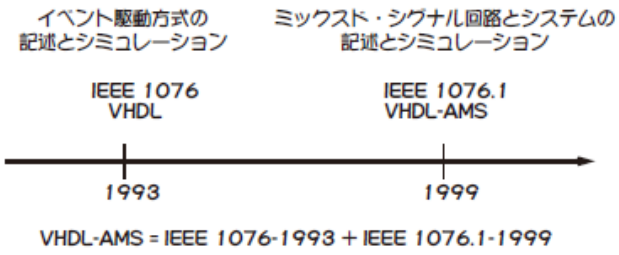


図 4-1 VHDL-AMS の歴史

2004 年，VHDL-AMS は IC や電気的なモデル記述に留まらず，様々な物理的な現象を記述するために拡張され，物理モデルを記述するための拡張と数学演算パッケージが拡張された(IEEE std. 1076.1.1)．また，2008 年にはメモリ管理や暗号化など，より運用に必要な機能が定義されている（IEEE std. 1076-2008）．これが一般に VHDL-200x と呼ばれてい

る版であり，浮動小数点パッケージや文字列型定義等の拡張が行われている．

VHDL-AMS は，その始まりがデジタル IC の記述を起点としているため，「同等の性質」を持つ「複数の部品」を配置した場合のばらつきを考慮したシミュレーションに適していると言えるが，データや関数（メソッド）の継承は構造体レベルであり，高度な運用には利用ノウハウが求められる．

4.1.2 物理量 QUANTITY と変数

VHDL-AMS では多様な変数の種類が用意されている．アナログ信号は QUANTITY として宣言され，アクロス変数・スルー変数に相当するものは特別にブランチ変数(Branch Quantity)と呼ばれ，シグナルフローとして入出力されるインターフェース変数(Interface Quantity）やその他の演算補助に利用されるフリー変数(Free Quantity)などに分類される．また，デジタル信号は SIGNAL として宣言され，アナログ変数とは全く異なる扱いになる．これらの変数は，さらに型定義を有し，real 型，integer 型，std_logic 型やネイチャーで定義された物理系に基づく型（FORCE 型，VOLTAGE 型　等々）を持って宣言される．ネイチャーはスルー変数アクロス変数等に適用される物理的な性質に応じて IEEE が提供するパッケージライブラリとして用意されており，作成したいモデルに応じて利用する．また，ユーザが独自のネイチャーを定義することも可能である．

VHDL-AMS ではこの型宣言を厳密に運用するよう定義しており，異なる型への代入や接続が制限されている．また，モデルの特性方程式の数と，これら定義した変数の数が一致するように定式化を行う必要がある．

4.1.3 ENTITY - ARCHITECTURE

１つの独立した部品は，エンティティ(ENTITY) 句とアーキテクチャ(ARCHITECTURE) 句の宣言より構成される．エンティティ句はモデルへの入出力を定義し，初期値や入出力変数，非因果的端子などから構成される．アーキテクチャ句は物理的な挙動を定義するセクションで，実際に微分代数方程式やモデルの動作を記述する．図 4-2 にモデル構造の模式図を示す．

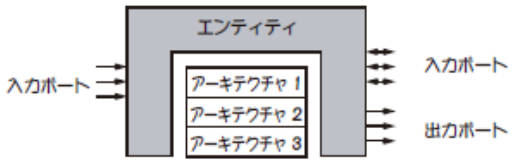


図 4-2　モデルの構造

VHDL-AMS では ENTITY 定義がそのまま１つの部品名の定義とし，　内部に複数の ARCHITECTURE を関連付けることが可能である．これは，状況により動作が大きく異なるようなモデル（例えば環境温度により挙動が全く異なる場合）を定義する際に活用され，より一般的にはモデルの詳細度を切り替えるために利用される．

```
1:  LIBRARY IEEE;
2:  USE IEEE.ELECTRICAL_SYSTEMS.ALL;
3:
4:  ENTITY R IS
5:      PORT (
6:          QUANTITY R : in RESISTANCE := 1.0e+3;
7:          TERMINAL p,m    : ELECTRICAL
8:      );
9:  END ENTITY R;
10:
11: ARCHITECTURE behav OF R IS
12:     QUANTITY v ACROSS i THROUGH p TO m;
13: BEGIN
14:     v == i*R;
15: END ARCHITECTURE behav;
```

図 4-3　VHDL-AMS による抵抗モデル

VHDL-AMS を用いて記述した抵抗モデルを図 4-3 に示す.

先頭に IEEE が提供する物理ライブラリへの参照を宣言し，本モデルが電気系非因果的端子を有することを示している．ENTITY～END ENTITY が部品名の宣言である．PORT(～)は端子の定義であり，TERMINAL にて電気の物理量に属する端子を宣言し，PORT( )句にて外部に非因果的接続可能な端子として定義している．同様に QUANTITY として物理量 R を入力変数として外部から入力できるように定義している．続く数値はデフォルト値である．この QUANTITY R は因果的接続されるインターフェース変数として扱われ，固定値ではない.

ARCHITECTURE ～ END ARCHITECTURE はモデル動作の定義句である．QUANTITY v ACROSS i THROUGH p TO m 行にて，電気端子 p, m 間のスルー変数・アクロス変数の関係を定義し，その上で BEGIN 以下に抵抗のモデル式として電圧方程式 $v = iR$ を記述している.

次にシグナルフローモデルの一例として VHDL-AMS を用いて記述した PI 制御モデルを図 4-4 に示す.

```
1:  ENTITY PI_CTRL IS
2:      GENERIC (
3:          kp : REAL := 1.0 ;
4:          ki : REAL := 1.0
5:      ) ;
6:      PORT (
7:          QUANTITY q_in    : in    REAL := 1.0 ;
8:          QUANTITY q_out : out REAL
9:      ) ;
10: END ENTITY PI_CTRL ;
11:
12: ARCHITECTURE behav of PI_CTRL IS
13: BEGIN
14:     q_out == kp * q_in + ki * q_in'integ ;
15: END ARCHITECTURE behav;
```

図 4-4　VHDL-AMS による PI 制御モデル

物理モデル(NATURE)に基づくモデルではなく，且つ入出力信号の型も特定する必要が無いため，先頭での IEEE パッケージ参照の宣言は不要である．ENTYTY～END ENTITY が部品を定義している．GENERIC (～) は部品の定数パラメータであり，kp, ki は固定値である．PORT( ) 句にて，外部からの因果的接続信号入力 q_in 及び，外部への因果的接続信号出力 q_out を定義している．

ARCHITECTURE ～ END ARCHITECTURE はモデル動作の定義句である．シグナルフローモデルのため，入出力信号の関係式 $q_{out} = Kp \cdot q_{in} + Ki \cdot \int q_{in} dt$ を直接記述している．ここで，q_in'integ は，物理量 q_in の時間積分を意味する．

4.1.4 PACKAGE とライブラリ

VHDL-AMS では複数のモデルから利用される定数や型の定義，関数等を一纏めにしたパッケージ（PACKAGE）句による定義が用意されている．図 4-5 にパッケージ宣言の例を示す．

```
1:  PACKAGE misc IS
2:      TYPE   short_int IS RANGE -100 TO 100 ;
3:      FUNCTION   bool2sint (b: BOOLEAN ) RETURN short_int ;
4:  END PACKAGE misc ;
5:
6:  PACKAGE BODY misc IS
7:      FUNCTION bool2sint (b : BOOLEAN) RETURN short_int IS
8:          VARIABLE sl : short_int :=0 ;
9:      BEGIN
10:         IF(b) THEN
11:             sl := 1 ;
12:         ELSE
13:             sl := 0 ;
14:         END IF ;
15:         RETURN sl ;
16:     END FUNCTION bool2sint ;:
17:
18: END PACKAGE BODY misc;
```

図 4-5　VHDL-AMS による PACKAGE 定義例

PACKAGE～END PACKAGE 句は宣言句である．特別な型 short_int や，bool2sint( ) 関数の宣言を定義している．関数の実体は PACKAGE_BODY～END PACKAGE_BODY 句に実装される．作成した関数は USE 文を指定した全ての部品より自由に呼び出し，利用することができる．

パッケージと部品をまとめた単位をライブラリと呼ぶ．ライブラリの模式図を図 4-6 に示す．

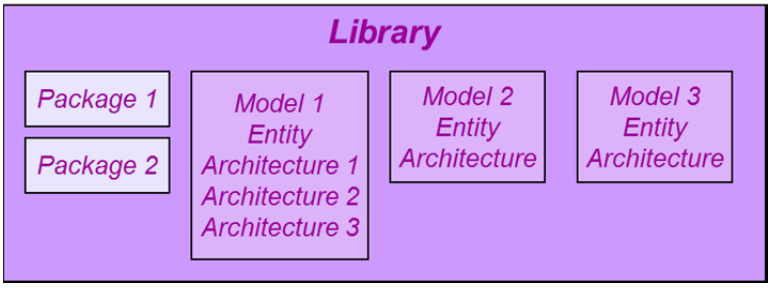


図 4-6　ライブラリ模式図

ライブラリの実装手段は環境を提供しているツールにより異なるが，ユーザは利用したい宣言が含まれるライブラリを“LIBRARY ライブラリ名; ”で指定し，また利用したいパッケージを“USE ライブラリ名.パッケージ名.ALL; ”等として指定する.

IEEE ではビット型宣言パッケージライブラリ(std_logic)や，数学関数ライブラリ(math_real)及び，物理ドメイン定義ライブラリ等を提供している．表 4-1 にその一部を示す．多くのデジタル信号処理パッケージに関しては表から省略している.

表 4-1　IEEE 提供のライブラリ（一部）

性質	
標準パッケージ	STD.STANDARD.ALL
ファイル入出力	STD.TEXT_IO.ALL
デジタル型処理	IEEE.STD_LOGIC_1164.ALL
数式処理	IEEE.MATH_REAL.ALL
物理定数	IEEE.FUNDAMENTAL_CONSTANS.ALL
電気	IEEE.ELECTRICAL_SYSTEMS.ALL
機械	IEEE.MECHANICAL_SYSTEMS.ALL
熱	IEEE.THERMAL_SYSTEMS.ALL
流体	IEEE.FLUIDIC_SYSTEMS.ALL
光学	IEEE.RADIANT_SYSTEMS.ALL
エネルギ	IEEE.ENERGY_SYSTEMS.ALL

モデル交換を視野に入れた場合，良く利用する関数（例えばマップ索引アルゴリズム）や，良く利用する基本要素部品（例えば抵抗やダンパ）などは，パッケージライブラリとして提供することが望ましい．この際異なる VHDL-AMS シミュレーション環境で利用できるように，ライブラリ名やパッケージ名は共通としなければならない.

4.1.5　ビヘイビアモデル（Behavor Modeling）

連立微分代数方程式を用いてモデルの挙動を定義したものをビヘイビアモデルと呼ぶ.

図 4-7 に DC モータのビヘイビアモデル定義の例を示す. 電気ドメインと機械ドメイン双方の特徴を有し，(4-1)式で示す電圧方程式と(4-2)式で示す運動方程式の関係を定義している.

$$v_a(t) = i_a(t) \cdot R_a + L_a \frac{di_a}{dt} + k_e \cdot \omega(t) \tag{4-1}$$

$$J \frac{d\omega(t)}{dt} = k_e \cdot i_a(t) + m_w(t) \tag{4-2}$$

$$N_{RPM} = 60/2\pi \cdot \omega(t) \tag{4-3}$$

```
 1:  LIBRARY IEEE ;
 2:  USE IEEE.ELECTRICAL_SYSTEMS.ALL ;
 3:  USE IEEE.MECHANICAL_SYSTEMS.ALL ;
 4:  USE IEEE.MATH_REAL.ALL ;
 5:
 6:  ENTITY DCM IS
 7:      GENERIC(
 8:          la     : INDUCTANCE :=0.0095 ; -- Armature Inductance [H]
 9:          ke     : MAGNETIC_FLUX :=0.544; -- Electromotive Force [Wb]
10:          j      : MOMENT_INERTIA := 0.004 -- Inertia of rotor [kgm2]
11:      ) ;
12:      PORT(
13:        TERMINAL n1, n2 : ELECTRICAL ;
14:        TERMINAL shaft : ROTATIONAL_V ;
15:        QUANTITY ra       : IN RESISTANCE :=1.2 ; -- Armature Resistance [Ohm]
16:        QUANTITY nrpm    : OUT real :=0.0 -- Rotational speed [rpm]
17:      ) ;
18:  END ENTITY DCM ;
19:
20:  ARCHITECTURE behav OF DCM IS
21:      QUANTITY v ACROSS i THROUGH N1 TO N2 ;
22:      QUANTITY omega ACROSS torque THROUGH shaft TO ROTATIONAL_V_REF ;
23:  BEGIN
24:      v == i*ra + la*i'dot + ke*omega ;
25:      j * omega'dot == ke*i + torque ;
26:      nrpm == omega *60.0/(2.0*MATH_PI) ;
27:  END ARCHITECTURE behav ;
```

図 4-7　ビヘイビアモデルを用いた DC モータモデル

ビヘイビアモデルの場合，一般に付随して利用されるパッケージは IEEE が提供する標準パッケージのみであるため，モデルの交換性が高い.

4.1.6　構造モデル(Structural Modeling)

モデルを構成する部品への参照と接続関係を用いて定義されたものを構造モデルと呼ぶ. 図 4-8 に先と同一の挙動を定義した DC モータの構造モデル定義の例を示す. ここではパッケージライブラリ spice2vhd 及び fundamentals_vda に定義されている要素部品（抵抗，

インダクタ，電圧源，イナーシャ，トルク源）らを利用し，その接続関係を PORT MAP (～) で, 入力パラメータへの代入を GENERIC MAP (～)を用いて関連付けしている. 一部 “emf == Ke * omega ; ”として逆起電圧と回転速度の代数式を定義しているが，このように構造モデル内に代数微分方程式を混在させることも可能である．一般に構造モデルは各ツールの GUI を用いて回路網を構築し VHDL-AMS モデルを出力することで容易に作成できる. 一方でモデル交換のためには利用しているパッケージライブラリの共有や要素部品のパラメータ定義型に留意して定義する必要が生じる. 例えば図 4-7 のビヘイビアモデルでは巻線抵抗 Ra を入力変数定義としたが，図 4-8 の構造モデルではエンティティ句にて定数定義としている．これは Ra を利用している下位部品 resistor1 にて定数（GENERIC）宣言されており，変数⇒定数への代入が制限されるためである.

```
1:  LIBRARY ieee;
2:  USE ieee.electrical_systems.all;
3:  USE ieee.mechanical_systems.all;
4:  USE ieee.fundamental_constants.all;
5:  USE ieee.math_real.all;
6:  LIBRARY spice2vhd;
7:  LIBRARY fundamentals_vda;
8:
9:  ENTITY DCMsc IS
10:     GENERIC(
11:         la     : INDUCTANCE :=0.0095 ; -- Armature Inductance [H]
12:         ke     : MAGNETIC_FLUX :=0.544;-- Electromotive Force [Wb]
13:         j      : MOMENT_INERTIA :=0.004;-- Inertia of rotor [kgm2]
14:         ra     : RESISTANCE :=1.2-- Armature Resistance [Ohm]
15:     ) ;
16:     PORT(
17:         TERMINAL n1, n2 : ELECTRICAL;
18:         TERMINAL shaft : ROTATIONAL;
19:         QUANTITY nrpm : OUT real :=0.0 -- Rotational speed [rpm]
20:     ) ;
21: END ENTITY DCMsc;
22:
23: ARCHITECTURE struct OF DCMsc IS
24:     TERMINAL net_53, net_52, net_51 : ELECTRICAL;
25:     TERMINAL net_3 : ROTATIONAL ;
26:     QUANTITY current, emf : real :=0.0;
27:     QUANTITY ang_rad, omega, torque : real :=0.0;
28: BEGIN
29:     resistor1 : ENTITY spice2vhd.resistor(spice)
30:                 GENERIC MAP ( r => Ra )
31:                 PORT MAP ( p => n1, n => net_53);
32:     inductor1 : ENTITY spice2vhd.inductor(spice)
33:                 GENERIC MAP ( l => La )
34:                 PORT MAP ( p => net_53, n => net_52 );
35:     am1 : ENTITY fundamentals_vda.current2q_sensor_vda(simple)
36:                 PORT MAP ( el_1 => net_52, el_2 => net_51, q_out => current );
37:     vsrc1 : ENTITY fundamentals_vda.q2voltage_vda(simple)
38:                 PORT MAP ( el_2 => n2, el_1 => net_51, q_in => emf );
39:     emf == Ke * omega ;
40:
41:     ang_sens1 : ENTITY fundamentals_vda.angle2q_sensor_vda(simple)
42:                   PORT MAP ( mr_2 => ROTATIONAL_REF, mr_1 => shaft, q_out => ang_rad );
43:     omega == ang_rad'dot ;
44:     rigid_body1 : ENTITY fundamentals_vda.rigid_body_rot_vda(basic)
45:                   GENERIC MAP ( j => j )
46:                   PORT MAP ( mr_2 => shaft, mr_1 => net_3);
47:     trqsrc1 : ENTITY fundamentals_vda.q2torque_mr_vda(simple)
48:               PORT MAP ( mr => net_3, q_in => torque );
49:     torque == Ke * current ;
50:     nrpm == omega *60.0/(2.0*MATH_PI) ;
51: END ARCHITECTURE struct;
```

図 4-8　構造モデルを用いた DC モータモデル

4.1.7　ミックスド・シグナル

VHDL-AMS ではアナログ信号とデジタル信号が 1 つのアーキテクチャ内で混在した系

(ミックスド・シグナル・システム）を定義でき，その応用範囲は広い．デジタル信号はイベントドリブンで処理されるが，アナログ信号の変化による状態の検知と割り込みの発生などに応用される．その一例として，図 4-9 に VDA が提供するライブラリのギャップ（隙間）モデル [7]を示す．端子間の距離が S_REL0 以下になった場合には高剛性 C で反発するスティフな系であり，安定化のために減衰係数 D を有する．距離が開いた場合には力の伝達は行わない．”CONTACT <= not S_REL'ABOVE(S_REL0);”にて接触判定を行い，”break on CONTACT;”で割り込みを掛けてアナログ部の再計算を行うように指示している．このようなデジタル割り込み処理はアナログモデルにおいても多用される手法であり，デジタル記述に関しても一連の仕組みを理解しておくとより高度なモデル記述に対応することができる.

```
 1:  LIBRARY IEEE ;
 2:  USE IEEE.MECHANICAL_SYSTEMS.ALL ;
 3:
 4:  ENTITY ELASTOGAP is
 5:      GENERIC(
 6:          S_REL0   : REAL      := 0.0;
 7:          C        : REAL      := 1.0;
 8:          D        : REAL      := 0.0
 9:      );
10:      PORT (
11:          TERMINAL FLANGE_A : TRANSLATIONAL;
12:          TERMINAL FLANGE_B : TRANSLATIONAL
13:      );
14:  END ENTITY ELASTOGAP;
15:
16:  ARCHITECTURE BASIC of ELASTOGAP is
17:      QUANTITY S_REL across F_B_A through FLANGE_B to FLANGE_A;
18:      SIGNALCONTACT : BOOLEAN := TRUE;
19:  BEGIN
20:
21:      CONTACT <= not S_REL'ABOVE(S_REL0);
22:
23:      if CONTACT use
24:          F_B_A == C*(S_REL - S_REL0) + D*S_REL'DOT;
25:      else
26:          F_B_A == 0.0;
27:      end use;
28:
29:    break on CONTACT;
30:  END ARCHITECTURE BASIC;
```

図 4-9　弾性ギャップ(ELASTOGAP)モデル

また，より高度なモデルの作成に当たっては文献 [8]，[9]を参照されたい.

4.1.8　モデル接続のための留意点

モデル接続を考慮したモデリングとして，先ず考慮すべきは入出力変数や端子の型定義である．事例で示したように VHDL-AMS では，各パラメータの単位系や意味に関して特

定する規定は無い．このためにコメント等を用いてより理解しやすい補足情報を付記することを推奨している．また，特に構造モデルの場合には利用している要素部品が格納されているパッケージライブラリの互換性も重要になる．構造モデルは GUI を用いて作成された回路モデルより作成されることが多いため，変数⇒定数の接続や浮き端子の処理などツールの回路編集機能特有の処理が行われている場合もあり，そのツールの特徴を把握しておく必要がある．さらに，連成シミュレーション等のツール連携を行う場合には，イベント処理による割り込みや巻き戻しといったミックスド・シグナル特有の現象に対し，被連携ツールの動作を確認しておくことも必要である．

4.2 Modelica

4.2.1 概要

Modelica は，微分代数方程式を利用したシステムモデリングのためのオブジェクト指向の言語である．言語仕様は Modelica Association がとりまとめている．1997 年に Ver. 1.0 がリリースされ，2014 年 10 月現在での最新版は Ver. 3.3 Revision1 である．

Modelica では，数式で記述された詳細なモデルをコンポーネントとしてライブラリ化し，それらを用いて高次のモデルを構築することで，過去に蓄積された知識を再利用したモデルの大規模化が実現されている．このように階層を構造化したモデリングは，モデルの「クラス」を導入してインスタンス化することで達成される．グラフィカル・エディタを使用してコネクタを結線することで，モデルコンポーネント間の物理的な接続を表し，継承により部品の容易な適用がサポートされている．Modelica ではオープンソースの Modelica Standard Library (MSL)が提供されており，そこには数学をはじめ，電気，磁気，力学，熱，流体の物理領域における 1280 の要素モデルと 910 の関数が含まれている．

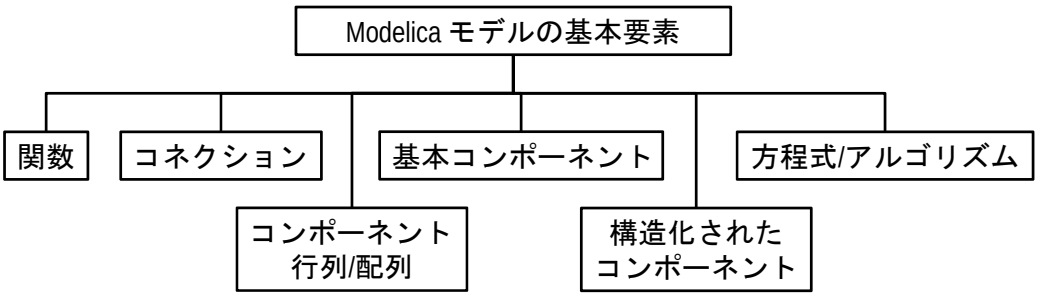


図 4-10 Modelica モデルの基本要素

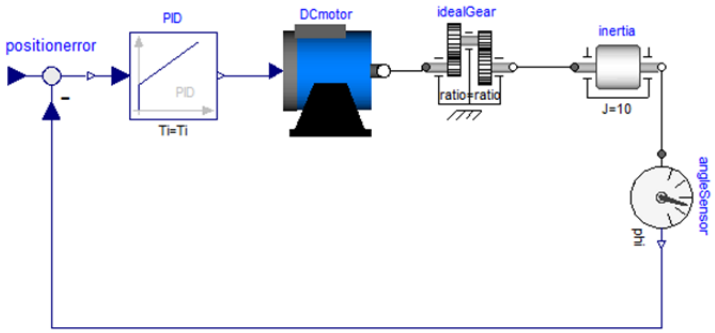
図 4-10 に，Modelica モデルの基本要素を示す．これらの概念は，モデルの階層構造化や再利用だけでなく，大規模で複雑なモデル開発をサポートするために，アプリケーションドメインや専門の領域に依存する表現とは独立に使用することができる．以下，本節で

は，Modelica に用いられる基本的概念とコンポーネントをつなぐ方法について述べる．

4.2.2 Modelica モデルの例

図 4-11 に角度を制御量として PID 制御を施した DC モータのモデルを示す．図 4-11(a)に示すように Modelica はアイコンを用いたグラフィカルユーザーインターフェースを持ち，図 4-11(a)のように記述されている．このモデルは，動的モデルである DC モータとギヤ，イナーシャ，フィードバック制御のための角度計，誤差検出器，PID 制御器から構成されている．なお，図 4-11(b)では，各行末のセミコロン(;)の手前に a が記載されているが，これは非表示にされたアノテーションを表している．これについては次項のモデルの項目で述べる．

図 4-12 に示した DC モータのように，部品のモデルもまた階層構造を持つ．DC モータのモデルは，指令値に対応した電圧を発生させる電圧源，抵抗，インダクタ，電流—トルク変換機およびイナーシャで構成されている．これらの部品はさらに階層構造をもっており，図 4-12 (b)でインダクタについて示すように，言語表記で確認することができる．

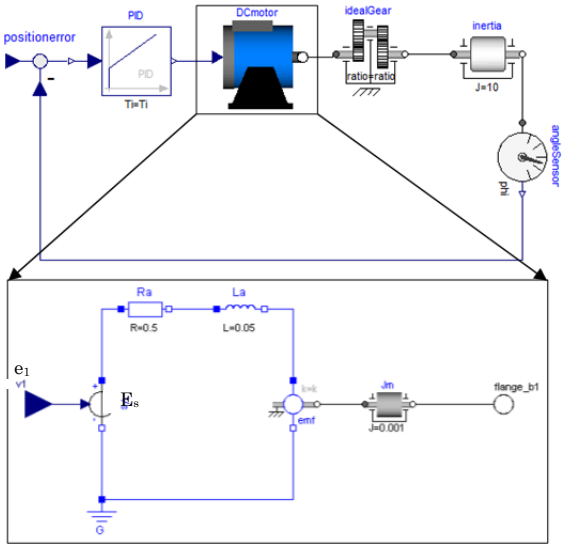


(a) グラフィカル表記

```
model MotorDrive
    dCmotor DCmotor                    a;
    Modelica.Blocks.Continuous.PID PID                 a;
    Modelica.Mechanics.Rotational.Components.IdealGear idealGear        a;
    Modelica.Mechanics.Rotational.Sensors.AngleSensor angleSensor       a;
    Modelica.Mechanics.Rotational.Components.Inertia inertia(J=10)      a;
    Modelica.Blocks.Math.Feedback positionerror        a;
equation
    connect(positionerror[1].y, PID.u)   a;
    connect(PID.y, DCmotor.' 1)          a;
    connect(dCmotor.flange_b, idealGear.flange_a)      a;
    connect(idealGear.flange_b, inertia.flange_a)      a;
    connect(inertia.flange_b, angleSensor.flange)      a;
    connect(angleSensor.phi, positionerror[1].u2)      a;
end MotorDrive;
```

(b) 言語表記

図 4-11 モータドライバモデルの構成



(a)グラフィカル表記

```
model DCmotor
  ... SignalVoltage     Es              a;
  ... Basic.Resistor    Rs (R=0.5)      a;
  ... Inductor          Ls (L=0.05)     a;
  ... Ground            G               a;
...
equation
  connect(e1, signalVoltage.v)          a;
  connect(signalVoltage.n, resistor.p)  a;
  connect(signalVoltage.p, ground.p)    a;
...
end DCmotor;
```

```
model Inductor
   parameter Modelica.SIunits.Inductance L;
   ... Interfaces.PositivePin              pin_p;
   ... Interfaces.NegativePin              pin_n;
equation
   0 = pin_p.i + pin_n.i;
   L*der(pin_p.i) = pin_p.v – pin_n.v;
end Inductor;
```

```
connector PositivePin
package SIunits = Modelica.SIunits;
SIunits.Voltage          v;
flow SIunits.Current     i;
end PositivePin;
```

```
type Voltage =Real(quantity="Voltage", unit ="V");
```

(b)言語表記

図 4-12　DC モータモデルの階層性

4.2.3 Modelica モデルの基本要素

・コンポーネント

Modelica モデルは，方程式による記述の他，コンポーネントを使用して記述することができる．コンポーネントは作成する Modelica モデルの中で次のように宣言することで使用が可能となる．

```
ComponentClass componentName(par1=value1, par2=value2);
```

ここでは，まず使用するコンポーネントが含まれるライブラリ名を，文頭の”ComponentClass”の箇所に指定する．続いて使用するコンポーネント名を”componentName”の位置で指定し，その後の括弧内では，パラメータのデフォルト値を変更するための，名前と値のペアのリストを指定する「修飾子」を付けることができる．

Modelica の基本コンポーネントは，次のように表される．

```
Real u, y(start=1);          Specifier – Real, Integer, Boolean, enumeration, etc…
parameter Real T=1;          predefined class or type – parameter, constant
```

コンポーネントは次のように構造化して記述することもできる．

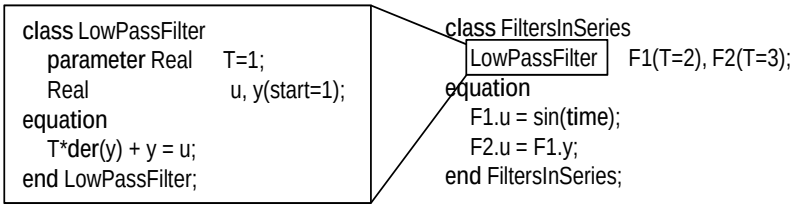


図 4-13 コンポーネントの記述

ここでは，右側の FilterInSeries というコンポーネントの中で，LowPassFilter というコンポーネントが使用されている．

・コネクション

Modelica のモデルはコンポーネントを接続して作成することができるが，接続（コネクション）のためには，個々のコンポーネントの端子（コネクタ）でポート間の相互作用の形式を定義する必要がある．この接続による相互作用の定義のための特殊なクラスを「connector」という．たとえば，下記の「connect」という名前の特殊なステートメントでは，component1 の connectorA と，component2 の connectorB の 2 つのコネクタ間の接続を定義している．

```
connect(component1.connectorA, component2.connectorB);
```

図 4-14 に示したモデルでは，equation 内の 2 つの connect 文で，それぞれ PID と DCmotor，DCmotor と idealGear の接続が記述される．

```
model MotorDrive
  dCmotor DCmotor                    a;
  Modelica.Blocks.Continuous.PID PID                    a;
  Modelica.Mechanics.Rotational.Components.IdealGear idealGear          a;
   ・ ・ ・ ・
equation
  connect(PID.y, DCmotor.ef)         a;
  connect(dCmotor.flange_b, idealGear.flange_a)         a;
   ・ ・ ・ ・
end MotorDrive;
```

図 4-14　connect によるコネクタ間の接続

・変数の型

Modelica には，組み込みの型として，Real (実数型)，Integer (整数型)，Boolean (ブール型)，String (文字列型)が用意され，列挙型を取り入れることも可能である．Real 型には，quantity (数量の名前)，unit (測定の単位)，displayUnit (表示単位)，min (最小値)，max (最大値)，start (開始値)，fixed (開始値が固定値か近似値か)のような一連の属性を持つ．物理モデリングでは物理的な数量の間にある関係を指定することが課題となるが，Modelica では次のような構文を使用してこれらの「型」を宣言する．

```
type NewType = TypeName(attr1=value1, attr2=value2);
```

例えばサーボシステムの場合，角度やトルクなどの数量が対象となる．

```
type Angle = Real(quantity="Angle", unit="rad", displayUnit="deg");
type Torque = Real(quantity="Torque", unit="N.m");
```

Modelica の使用を簡素化し互換性を維持するために，幅広い型定義の標準ライブラリが存在しており，Modelica トランスレータとともに提供されている．

・変数

変数はコンポーネントと同様の方法で宣言される．プレフィックスには，例えば，**parameter**，**input**，**output**，**flow** を指定できる．

```
prefix VariableType variableName(attr1=value1, attr2=value2);
```

・方程式

Modelica の方程式は，式中に記述された変数や関数などが，等号で結ばれた両辺で等しいことを示している．等号の左辺の変数に右辺を代入するものではない．式では，通常の数学演算子，関数に対する呼び出し，if-then-else コンストラクトが使用される．特殊演算子 der()は時間微分を示すために使用され，time は時間を表すために使用される．方程式は以下のような形で記述される．

```
equation
T*der(u) = y;
```

・コネクタ

図 4-15 に回転力学コンポーネントのコネクタの記述例を示す．コンポーネント間の相互作用を記述するために使用される変数はコネクタ内で宣言される．この例では角度 phi とトルク tau である．

```
connector Flange "1-dim. rotational flange of a shaft"
  SI.Angle phi "Absolute rotation angle of flange";
  flow SI.Torque tau "Cut torque in the flange";
end Flange;
```

図 4-15　コンポーネント間の相互作用の記述

回転力学コンポーネントの 2 本のシャフトが接続される場合には，角度が等しいように制約され，さらにトルクの合計は 0 となる．図 4-15 中でプレフィックスとして flow が付いた変数 tau は，接続されたコネクタすべてに関して合計が 0 となり，3.5 節で説明したスルー変数を定義する場合に使用される．例えば，電気回路の電流（キルヒホフの電流則）や，流体モデルの質量流量にも使用される．機械システムに取り入れたスルー変数は，フリーボディダイアグラム（自由体図）に記載される力やトルクに対応している．

・モデル

図 4-16 に Modelica モデルの例を示す．

```
model Inertia "1D-rotational component with inertia"
  Flange flange_a "Left flange of shaft";
  Flange flange_b "Right flange of shaft";
parameter SI.Inertia J(min=0) "Moment of inertia";
  SI.AngularVelocity w "Absolute angular velocity of component (= der(phi))";
equation
  flange_a.phi = flange_b.phi;
  w = der(flange_a.phi);
  J*der(w) = flange_a.tau + flange_b.tau;
end Inertia;
```

図 4-16　変数とコネクタ，方程式の記述

詳細部品のモデルを組み合わせて作られたモデルには，通常，変数とコネクタの宣言のセットと方程式のセットが含まれる．再利用可能な記述を作成するため，「拡張」できる部分モデルを定義し，再利用する機能を有している．

Modelica の表現には「アノテーション」が含まれており，これには例えば，グラフィカルなアイコン表現や，接続の経路制御の方法がある．これらのアノテーションはエディタ上で表示・非表示を選択でき，通常は非表示になっている．

・Modelica モデルのトランスレート

x を状態量，w を代数変数，p をパラメータとするとき，Modelica の数学モデルは，次式のようなハイブリッド型の微分代数方程式（DAE)として表現される．

$$F\left(\frac{dx}{dt}, x, w, p\right) = 0 \tag{4-4}$$

このため Modelica では，3.6 節で述べられた非因果的なモデリングが行われる．シミュレーションを実行する際には，コンピュータ計算を行いやすくするために，多くの場合次式のような常微分方程式（ODE)にトランスレートを行っている．

$$\frac{dx}{dt} = f(x, p) \tag{4-5}$$

このような変換を施すことで代数的に w を解くためのコードが生成される．このため数値的に w を解く必要がない．トランスレートのプロセスでは，モデルの構造的な解析とシンボリック処理が行われる．

・Modelica のより高度な記述方法

変数やコンポーネントには，多次元配列を取り入れることができる．式内ではベクトル操作や「行列操作」を使用できる．方程式は通常，並び替えられた後に解が求められる．モデル内や Modelica 関数内に「アルゴリズム」を定義することも可能である．そうしたアルゴリズムでは，for ループや while ループが使用できる．「演算子オーバーロード」を行うことで，複合操作の定義や多項式に対する操作の定義も可能になる．コンポーネントの ComponentClass を「再宣言」することで，あるコンポーネントに対してどのモデルを使用するかを変更することもできる．特殊な when コンストラクトを使用すると，時間的に特定の瞬間にのみ有効な「瞬間方程式」を取り入れることができる．そうした方程式の「同期」に関する特別な検討事項については，文献 [10]を参照されたい．

・Modelica に関する参考文献

Modelica に関する参考文献については，以下の Modelica Association のホームページで，代表的な書籍とチュートリアルが，解説とともに紹介されている．

https://www.modelica.org/publications

第5章　FMI と FMU 概要

FMI は Functional Mock-up Interface の，FMU は Functional Mock-up Unit のそれぞれ頭文字を取った略語である．規格である FMI に基づいて作成された実行可能なファイルを FMU と呼ぶ.

5.1　経緯

5.1.1　MODELISAR プロジェクト

2008 年から 2011 年の期間で Information Technology for European Advancement(ITEA2)プロジェクトのサブプロジェクトとして MODELISAR と呼ばれるプロジェクトが実施された．MODELISAR プロジェクトはドイツ・ダイムラー社（DaimlarAG）が提案しオーストリア，ベルギー，フランス，ドイツ，スウェーデンの 5 か国から 29 の企業や団体が参加し，2670 万ユーロ，176 人年が投入された．CAD における DMU（Digital Mockup）が生んだ生産性向上をさらに進化させ，システム設計におけるモデリングとシミュレーションを実現するための規格を作ることが同プロジェクトの目的であった.
その成果として FMI for Model Exchange ver. 1.0（2010 年 1 月 26 日），FMI for Co-Simulation ver. 1.0（2010 年 10 月 12 日），FMI for PLM ver. 1.0（2011 年 3 月 31 日）が規定された．なお以降では FMI for Model Exchange を ME，FMI for Co-Simulation を CS と記述する.

5.1.2　MODELISAR プロジェクト以降

MODELISAR プロジェクト終了後，FMI の規格については Modelica Association（以下 Modelica 協会）の中のプロジェクトとして継続して審議・検討が行われている．MODELISAR プロジェクトで規定された ver.1.0 では ME と CS が個別に規定されていたが，これらを一元化し，さらに ver.1.0 では不足していた項目を付加した規格として ver.2.0 が 2014 年 7 月 25 日規定された.

FMI1.0 と FMI2.0 の間には互換性がない．このため，利用するツールにより FMI1.0 のみ，FMI2.0 のみ，FMI1.0 と 2.0 の双方対応など対応範囲に違いがあるので注意を要する．最新状況は以下の Modelica 協会プロジェクト WEB サイトに掲載されている.
https://www.fmi-standard.org/

5.1.3　FMI の目的

FMI を規定する上で主眼とされたことは

① シミュレーション用の動的モデル交換（Model Exchange）と連成（Co-Simulation）の標準的インターフェース規格とする
② モデリング言語に関わらず利用できる
③ モデルの知財（IP）や製品のノウハウを保護できる

である．①～③の目的と成果については（9th International Modelica Conference 2012, 10th International Modelica Conference 2014 などで）欧州の有力な OEM からその取組みが報告されていることもあり，各種のツール開発企業・団体が FMI への対応を発表している．対応ツールについては以下の WEB サイトに，互換性テスト（compatibility test）結果と共に掲載されている．
https://www.fmi-standard.org/tools

5.2 FMI による規定項目

5.2.1 FMI の種類

MODELISAR 以来 FMI は前述のように主に ME と CS に関して規定されてきた．いずれも計算機シミュレーションを実行するための規格であるが，ME は同一の積分器（ソルバ）を用いる方式であるのに対して CS は FMU 内に個別のソルバを持ち使用する点が最大の違いである．

https://trac.fmi-standard.org/export/700/branches/public/docs/Modelica2011/The_Functional_Mockup_Interface.pdf より引用

図 5-1 ME（左）と CS（右）の違い [11]

ME の場合 FMU は，図 5-1 中の Tool 側から与えられた入力信号と状態変数から状態変数の時間微分値を計算し Tool 側へ渡す．Tool はこれを積分して FMU に戻す役割を担う．ME では積分計算における誤差が収束するまで Tool と ME 間の繰り返し演算が行われる．一方 CS では Tool 側から与えられた現在時刻（t_i）における入力信号から，与えられた次の時刻（t_i+hc_i）の値を計算し Tool 側へ返す．この際返される信号は出力変数だけで状態変数を全て返すのではない．CS における入出力間の時間（hc_i）は，communication step size（以下：「通信間隔」）と呼ばれ，ver.1.0 では通信間隔は一定値とされていたが ver.2.0 から可変にすることができるようになった．

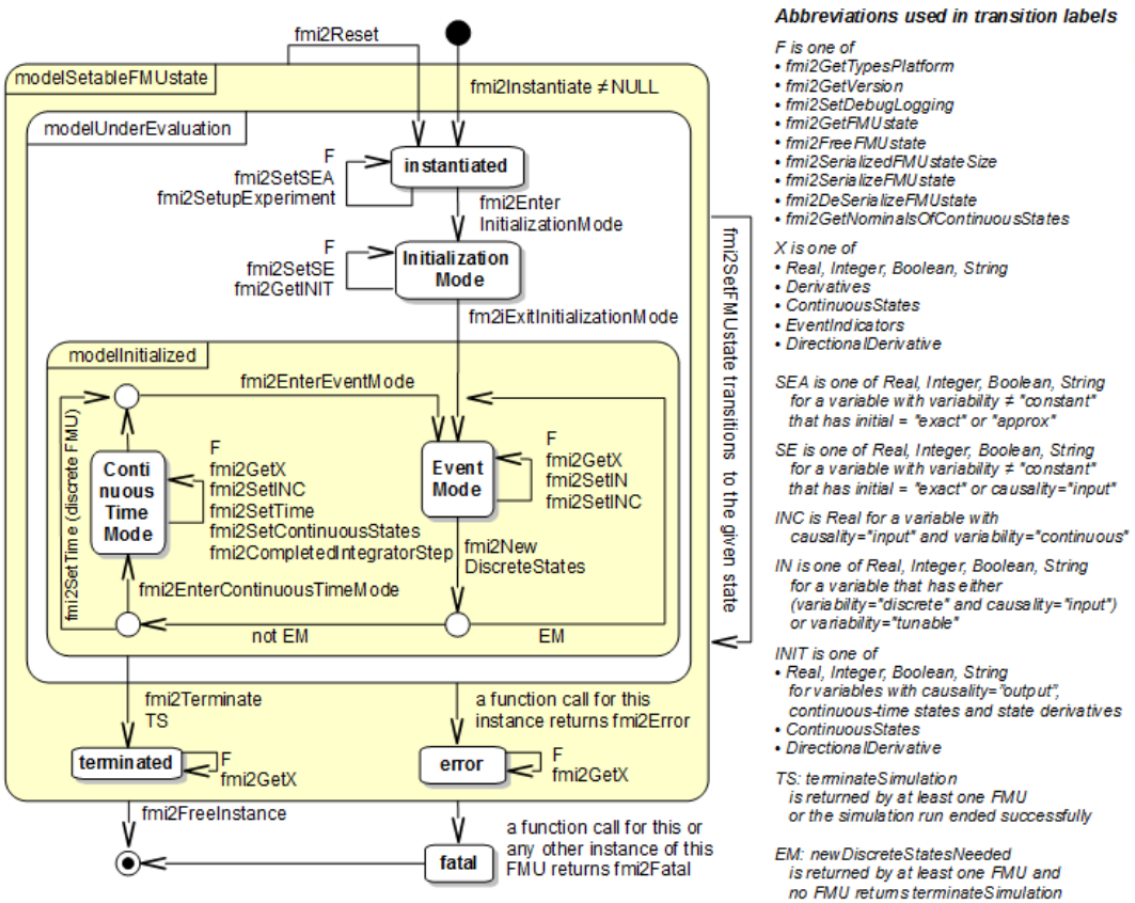
なお CS には分散計算環境を意識した，入出力データの橋渡しのみを FMU が行い，FMU とは別にモデルデータ及びソルバを持つモードも規定されているがここでは詳細は省く．

FMI 規格の中では，ME の FMU を作成する機能を Export 機能，読込み実行する機能を Import 機能，CS の FMU を作成する機能を Slave（スレーブ）作成機能，読込み実行する機能を Master（マスタ）機能と区別している．5.1.3 項で述べた WEB サイトの対応状況表では，上記の 4 つの機能に基づいて分類が行われている．

5.2.2 実行手順

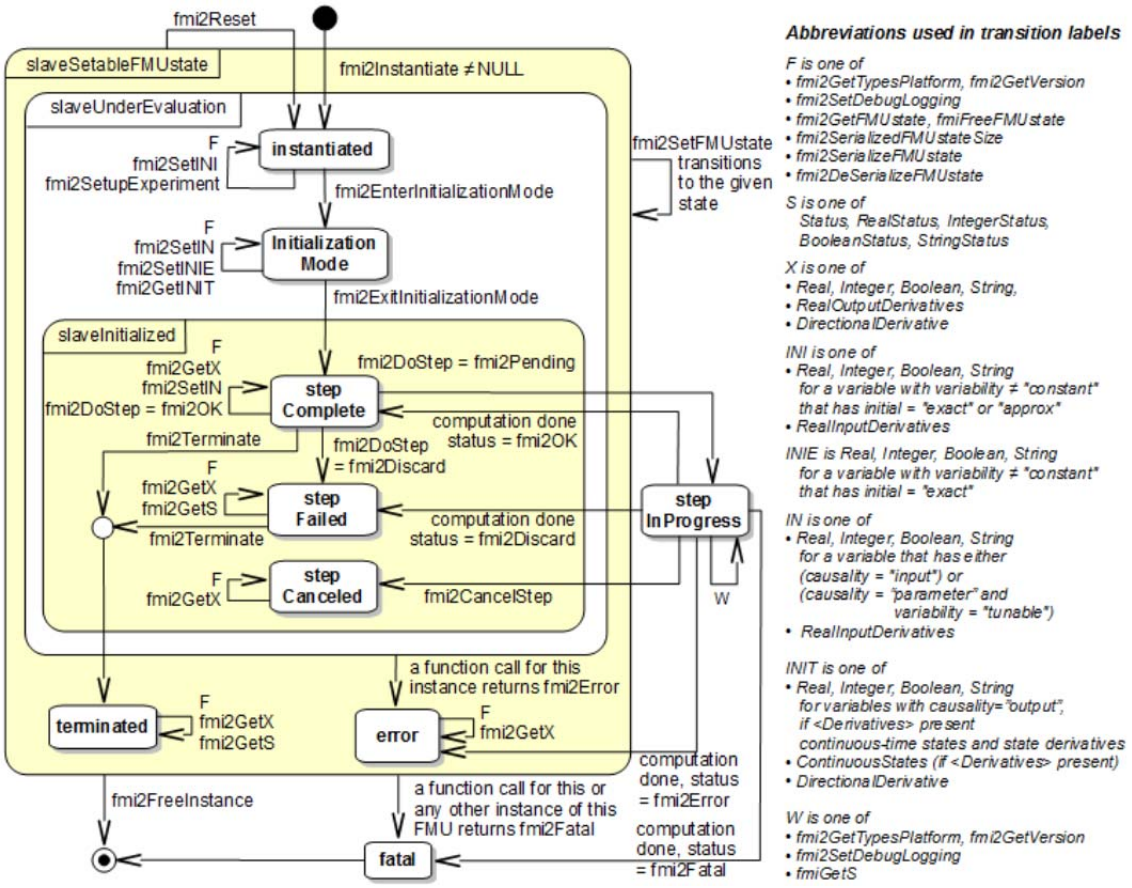
FMU を実行するためには実行環境が必要である．実行環境は各企業・団体から提供されている．5.2.1 項で述べた ME についての Import 機能を持つツールと CS についての Master がこの実行環境に相当する．実行環境ツールが FMU について実行を行う手順については FMI 規格の中で定められている．

なお前述の FMI プロジェクト WEB サイトから Compliance Checker が提供されており，FMU のチェックの一環として単体の FMU について実行させることが可能である．実行にはツール側からの呼び出しが必要であり，ME の場合と CS の場合で手順及び呼び出しに使用する関数が一部異なる．呼び出し手順と呼び出し時に使用する関数は ver.2.0 では ME の場合図 5-2，CS の場合図 5-3 のような手順・関数となっている．この手順や内部で使用される関数は，FMU を接続するツールを開発する立場では知る必要があるが，利用者の立場としては特に知る必要はない．



https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf より引用

図 5-2 Model Exchange の呼び出し手順（FMI2.0） [12]



https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf より引用

図 5-3　Co-Simulation の呼び出し手順（FMI2.0）［12］

5.2.3　FMU 作成

通常，各シミュレーションツールを開発している企業・団体が FMU の作成ツールも合わせて提供している. 作成される FMU は前項に掲載した図に示す各種の呼び出し関数で実行できるように，FMI 規格と共に提供されている各種の C ヘッダーファイルなどとコンパイルされ，Windows 系では DLL，Linux 系では SO 形式のファイルとなる.
また FMU には次節で説明する関連ファイル群も合わせて内包されるため，作成の過程では，この関連ファイルも自動生成またはマニュアル操作による追加が行われる.

なお，前掲の WEB サイト（https://www.fmi-standard.org/）で提供されている FMUSDK（Software Development Kit）を用いて，C コードから FMU を作成することも可能である.

5.3 FMU の構造

作成された FMU は ZIP 圧縮されたファイルで，拡張子として fmu を与える．ファイル名としては，例えば filename.fmu となる．拡張子を zip に変え，filename.zip とすると一般の解凍ツールを用いて unzip することができ内部構造を確認できる.

5.3.1 内部構造

FMU の内部構造を図 5-4 に示す．内部には 2 つのファイル（modelDescription.xml 及び model.png）と 4 つのフォルダー（documentation，source，binaries，resources）が存在する．modelDescription.xml は必須のファイルで，FMU の各種情報を含むファイルである．一方 model.png はモデルを表す画像ファイルで使用は任意である.
documentation フォルダーは FMU を説明するためのファイル群を保存するフォルダーである. HTML 形式で保存されるものとされており_main.html が標準のファイル名とされていてこの HTML ファイルからリンクを張ることが認められている．sources フォルダーはモデルのソースコードを入れるフォルダーで，resources フォルダーは，FMU で使用するデータなどを入れる．ただしこれらの使用は任意である．使用しなくても支障はない．一方で binaries フォルダーには，win32，win64，linux32，linux64 の 4 つのサブフォルダーを持つことが可能で，最低でも 1 つのサブフォルダーの中に実行形式のプログラムライブラリ（Windows 系では DLL, Linux 系では SO）を格納しなければならない. なお binaries フォルダー内の各サブフォルダーには ME，CS の一方または双方を格納することが可能である.

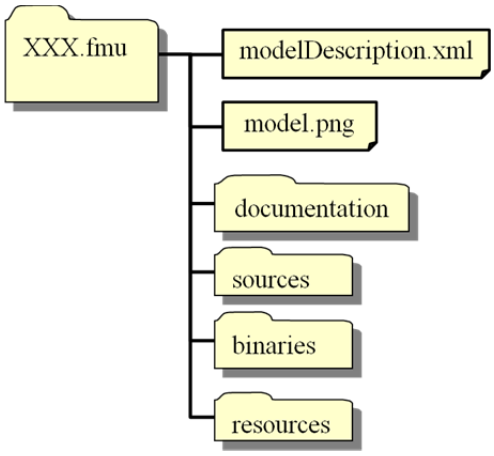
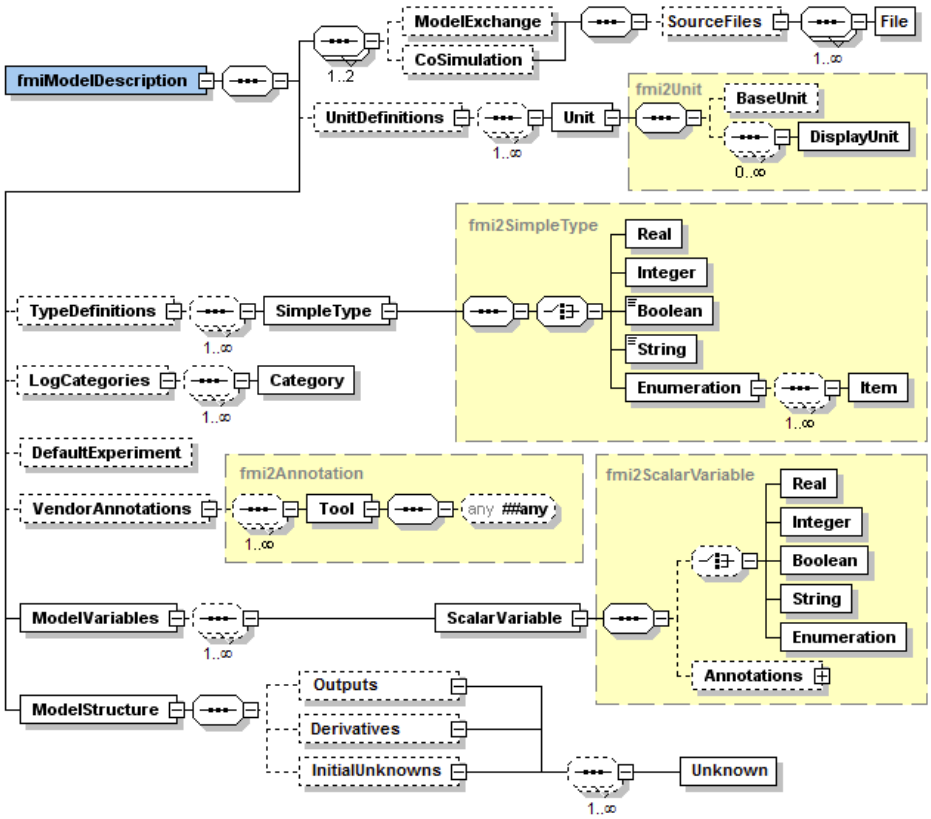


図 5-4 FMU の内部構造

5.3.2 xml による記述内容

FMU の中で一般ユーザが比較的確認しやすい部分は modelDescription.xml である．この xml のスキーマである xsd ファイルの構造を図 5-5 に示す．ただし全項目が必須であるわけではない．代表的な項目を表 5-1，表 5-2 に示す．なお FMI2.0 では FMI1.0 から変更

になっている部分があるため，参照のため 1.0 での内容も併記する．



https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf より引用

図 5-5　FMI ver.2.0 における modelDescription.xsd (FMI2.0) [12]

表 5-1　modelDescription.xml の項目（最上位）

項目	1.0	2.0	説明
（Attributes 参照）	必須	必須	モデルの概要情報
ModelExchange	なし	最低一方必須	ME の場合記述（内部に ModelIdentifier を持つ）
CoSimulation	なし		CS の場合記述（内部に ModelIdentifier を持つ）
UnitDefinitions	任意	任意	単位，表示単位の定義
TypeDefinitions	任意	任意	変数のタイプの定義
LogCategories	なし	任意	設定可能なログ情報のリスト
DefaultExperiments	任意	任意	デフォルトの解析設定値（終了時間など）
VendorAnnotations	任意	任意	ベンダーごとの注記
ModelVariables	任意	必須	変数の定義（1.0 では入出力などの区分）
Implementation	任意	廃止	1.0 の CS の際に使用する CS の区分
ModelStructure	なし	必須	変数の入出力種別と依存性説明

modelDescription.xml ファイルは可読な文字列で記述されているファイルであり，ツールを用いずともこのファイルからFMUの情報を得ることができる.例えばModelVariables, ModelStructure を読み取ることで，内部で使用されている変数名と，その変数が入力なのか出力なのか，どの変数から算出されるのかなどを知ることができる．また Attributes の部分を読むことでその FMU がどの FMI バージョンであるか，作成したツールが何かなどを読み取ることができる.
他の FMU と誤用されたり改変されたりしないように，FMU は guid を持っている．guid はユニークな数字であり，コンパイル時に FMU 中の dll にも書き込まれる．これにより FMU 作成時のプログラム以外が modelDescription.xml を書き換えて何らかの意図を持って guid を変更したとしても，dll と xml 中の guid が一致しないとエラー，実行不可とすることで誤用や改変に対する信頼性を担保している.

表 5-2　modelDescription.xml 最上位 Attributes として記載可能な項目（一部略）

名称	1.0	2.0	説明
fmiVersion	必須	必須	FMI のバージョン
modelName	必須	必須	モデル名
modelIdenifier	必須	移動	モデル識別用文字列 2.0ModelExchange または CoSimulation に移動した
guid	必須	必須	実行形式内の番号と一致するユニークな識別番号（global unique identifier）
description	任意	任意	簡易説明
author	任意	任意	作成者
version	任意	任意	バージョン
generationTool	任意	任意	作成ツール
generationDateAndTime	任意	任意	作成日時
variableNamingConventions	任意	任意	変数名の階層化有無（structured だと"."区切りの変数名が許容される）
numberOfContinuousStates	必須	廃止	連続状態変数の数
numberOfEventIndicators	必須	必須	イベントインジケータの数（ME のみ）

第6章　FMU 活用ガイドライン

6.1　モデル分割の考え方

モデルのサブシステム化・モジュール化は本来，機能配分の明確化やモデルの再利用の向上を目的として行われるが，本ガイドラインが対象としている「モデル接続技術」は企業内の複数の部門，あるいは複数の企業の担当者が作成したサブシステムモデルを接続・統合してシステムモデルあるいは試作車モデルを構築するためのものである．すなわち，ここで述べる FMI の使用を前提としたモデル分割は，例えば自動車メーカがバッテリメーカにモデルの仕様を提示して供給を依頼し，供給されたモデルを車両モデルに組み込んで使用するといった場合を想定している．よって，モデル分割の境界・粒度は業務プロセスやあらかじめ合意されたモデルアーキテクチャから決定されるものである．本ガイドラインの論旨は「モデル接続技術」であるため，分割の粒度に関する指針は述べない．

6.2　分割時のインターフェース設定の仕方

インターフェースを考えるにあたり，用語を定義する．まず，電気回路の端子のように，モデル化対象の「実際の」端子を「物理端子」と呼ぶ．次に，非因果的モデリングによりモデル化すると，入力・出力を明示しない端子が使用されるが，これを「非因果的端子」と呼ぶ．一方，因果的モデリングによりモデル化した際に使用される，入力・出力を明示する端子を「因果的端子」と呼ぶ．

FMI は，因果的接続が基本であるが，Modelica や VHDL-AMS などの非因果的モデリング言語によって作成されたモデルでも，FMI を用いて接続できれば，モデル活用範囲が拡大できるため，ここでは「非因果的モデリングで作成されたモデルを因果的接続する」際のインターフェースについて考える．

3.5 節で述べた通り，物理モデリングではスルー変数・アクロス変数の対をインターフェースに用いるのが一般的であり，非因果的モデリングではモデル化対象の物理端子ごとに非因果的端子を定義し，それぞれの物理量対を定義する（図 6-1）．このようなモデルを因果的接続する際には，もとの非因果的端子のスルー変数・アクロス変数を入出力変数として使用することが，汎用性が高く望ましい（図 6-2）．以下では，この考えに基づいて，インターフェースの考え方について述べる．

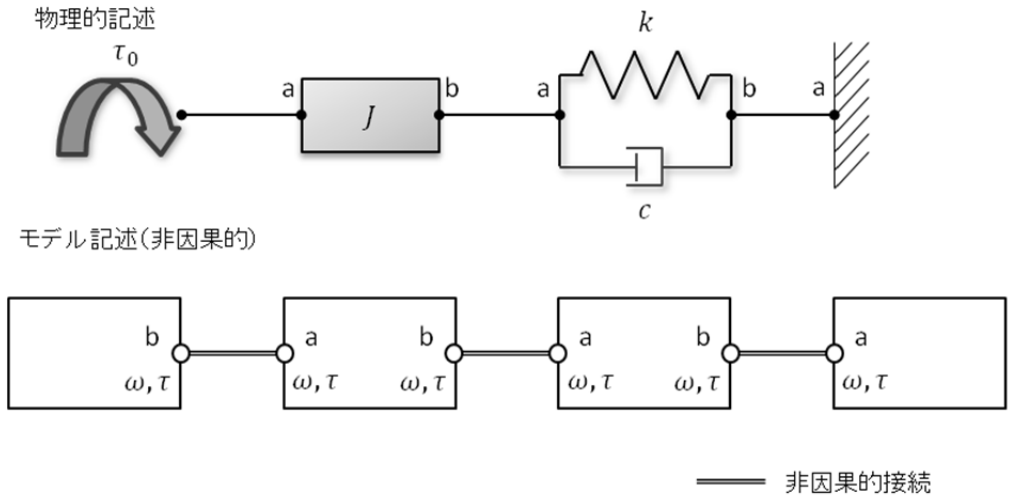


図 6-1 非因果的接続によるモデル表現

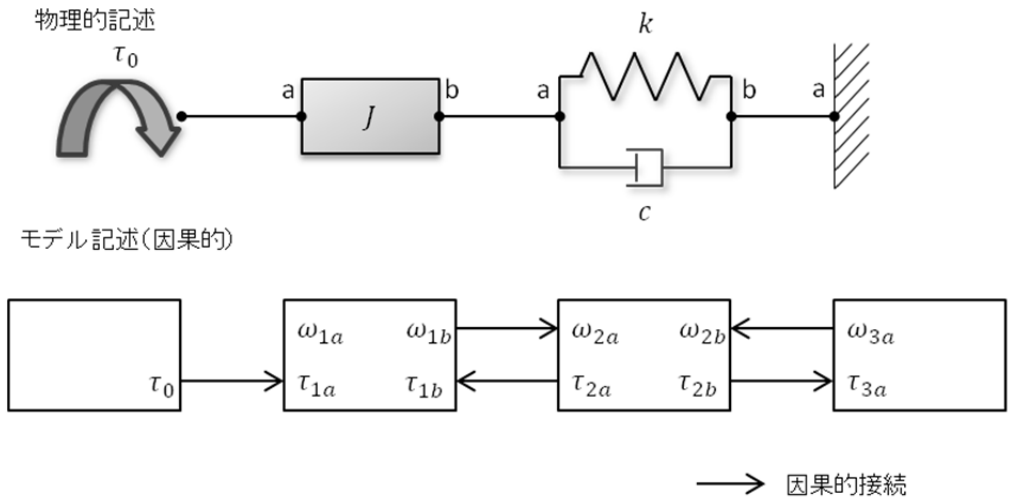


図 6-2 因果的接続によるモデル表現

6.2.1 信号極性の考え方

6.2.1.1 スルー変数の極性

スルー変数を入出力変数に持つモデルを因果的接続する際には必ず「信号極性問題」が生じるため，その都度，接続先のモデルを意識した極性定義を検討する必要がある（3.6.3 項）．ここでは，そのような意識をせずとも不整合が生じない，一貫した極性の定義方法について述べる．

まず，「接続する因果的端子同士は，一方の入力変数が他方の出力変数となっている」こ

とを前提条件とする．すなわち，一方のモデルが電流を出力とするのであれば，接続先のモデルでは電流を入力とする．これが成立していないと，それらのモデルは因果的接続が不可能であることを意味するため，妥当な条件であると言える．

3.6.3 項で述べた通り，2 つの因果的端子を接続する場合，各端子が示すスルー変数の値は等しくなるが，「物理端子」を考えてみると，一方は「流出」，他方は「流入」が起こっており，極性（符号の意味）が反転している（図 6-3）．これを回避するため本ガイドラインでは，意味の反転を認め，**「スルー変数を出力する因果的端子は正の値を『流出量』として扱い，スルー変数を入力される因果的端子は正の値を『流入量』として扱うことと定義する**（表 6-1）．上記の因果的端子間の入出力に関する前提と合わせて考えると，2 モデル（2 端子）間の接続では，必ず「一方の流入量＝他方の流出量」となり，矛盾のない定義となる．また，分岐や合流があった場合においても，「あるモデルの流入量＝複数のモデルの流出量の総和」（電流出力の抵抗モデル並列接続）や，「各々のモデルの流入量＝ある特定のモデルの流出量」（電流入力の抵抗モデルの直列接続）となり，整合性が保たれていることがわかる（図 6-4，図 6-5）．

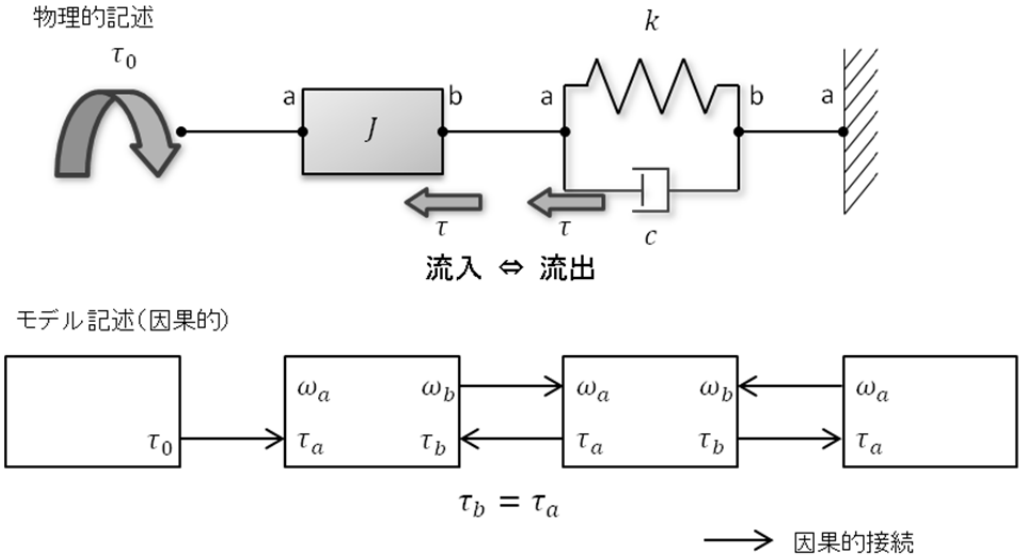


図 6-3 因果的接続時のスルー変数の極性の反転

表 6-1 スルー変数の極性と意味

	正のスルー変数	負のスルー変数
スルー変数を出力とするモデル	流出	流入
スルー変数を入力とするモデル	流入	流出

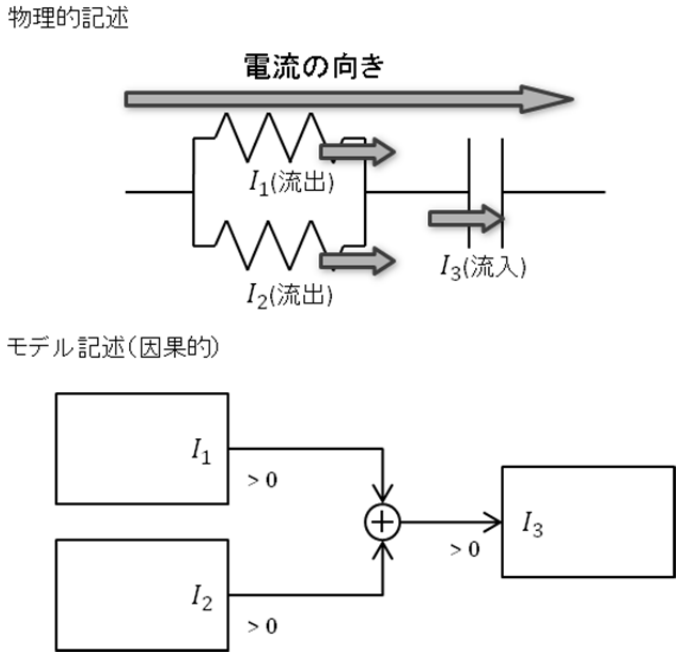


図 6-4 スルー変数の極性定義に則ったモデル接続の例 1

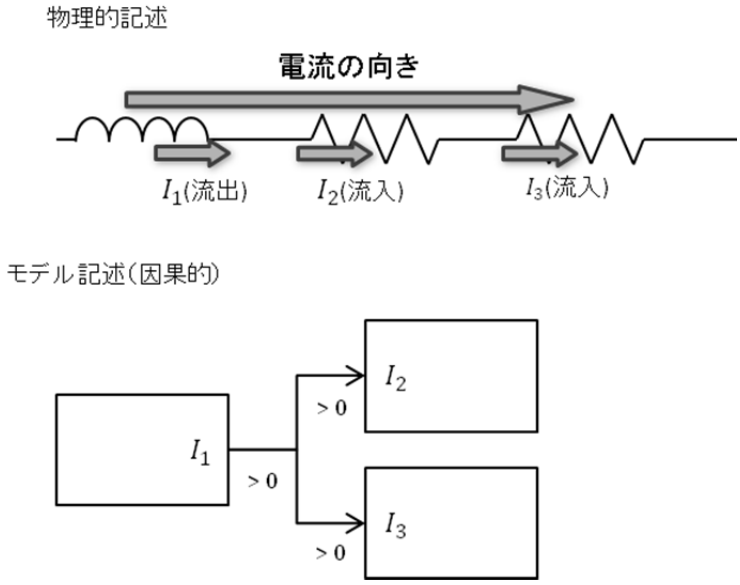


図 6-5 スルー変数の極性定義に則ったモデル接続の例 2

6.2.1.2 アクロス変数の極性

アクロス変数（速度・角速度・電圧・圧力など）については，スルー変数のように，2 つのモデルを接続した際に，双方で解釈が異なるということは生じない．あるモデルの出力

端子の電圧値が E [V]であれば，それを入力として受け取る側の端子も E [V]として受け取って問題ない．ただし，機構解析モデルなどのように 3 次元空間内に配置されたモデルの速度や変位についてはあらかじめモデルを交換する当事者間で座標軸の取り方について合意をしておく必要がある．本ガイドラインでは，車両運動のモデルであれば「自動車技術ハンドブック」の定義に従い，「x：車両前方，y：車両左方，z：車両上方」という座標系を採用することを推奨する．

6.2.2 FMU のインターフェースに対する考え方

FMI では因果的接続が基本であるが，ここで述べる「アダプタ」を使用すれば非因果的モデリングによって作成されたモデル（非因果的モデル）であっても，FMI によって接続することが可能である．ここではその考え方を，図 6-6 に示す「RL-CG 回路」を例に説明する．

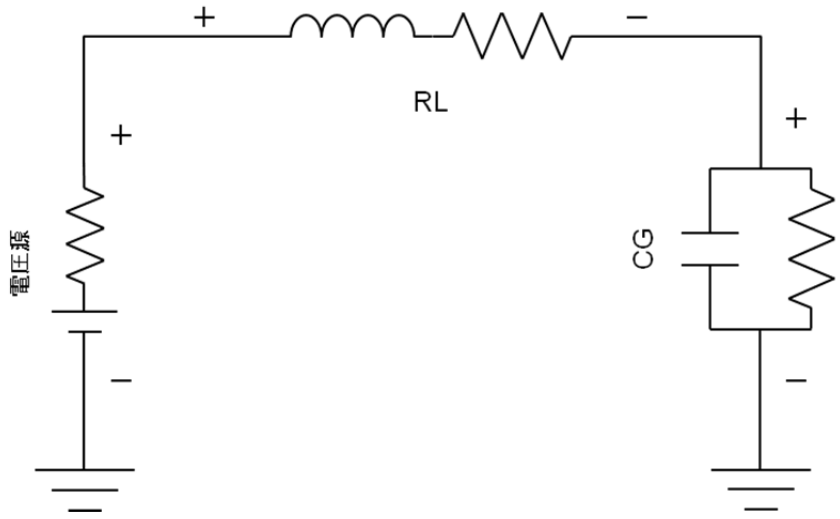


図 6-6 サンプル電気回路（RL-CG 回路）

まず，各モデル（電圧源，RL，CG）の入出力変数を表 6-2 および図 6-7 のように定義する．

表 6-2 サンプル電気回路における各モデルの入出力変数

モデル要素		入力変数	出力変数
電圧源	正極	電流	電圧
	負極	電圧	（電流）※
RL	正極	電圧	電流
	負極	電圧	電流
CG	正極	電流	電圧
	負極	電圧	（電流）※

※接地電極への電流は使用されない

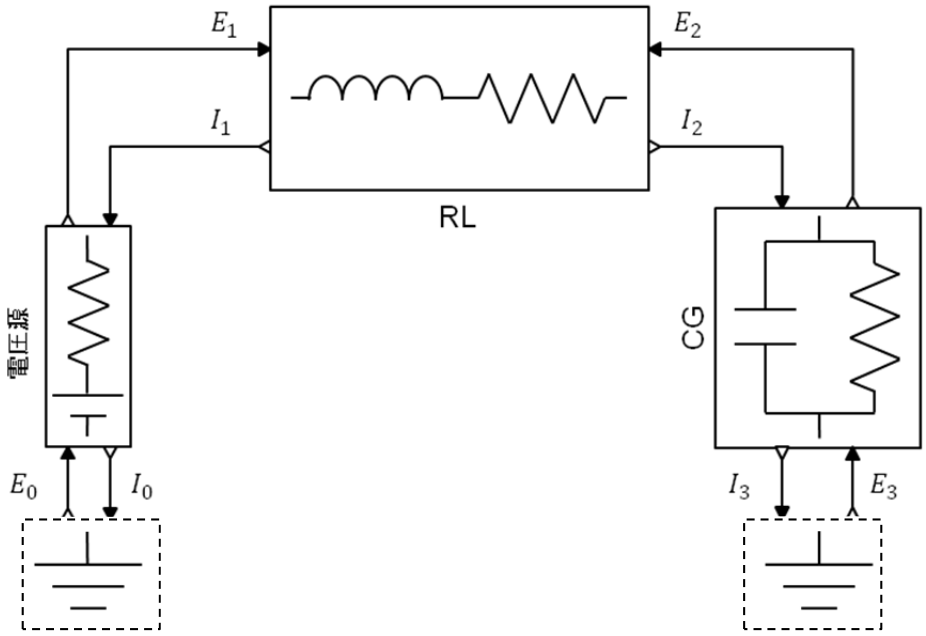


図 6-7　各モデルの入出力変数の対応関係

図中の四角で囲んだ部分が FMU 化の対象であり，内部のモデルを非因果的モデリングで作成することを考える．

Modelica では通常，スルー変数（flow 変数）は全ての端子において，流入する量を正の値として扱うので，ここでも各非因果的端子のスルー変数は流入量を正と定義する．一方で，各因果的端子の入出力変数の極性は，6.2.1 項で述べた信号極性の定義に従うようにモデルを記述すると，電圧源モデルの各端子のスルー変数の極性は図 6-8 の I_0 および I_1 ようになる．

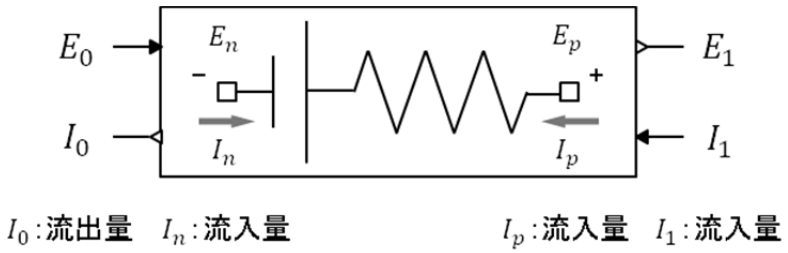


図 6-8　電圧源モデルの各端子の極性

このとき，図の－端子のように電流（スルー変数）を出力する端子で正負の意味（極性）が反転するため，

$$I_0 = -I_n \tag{6-1}$$

とすべきである．一方で，＋端子では極性は反転しないため，

$$I_1 = I_p \tag{6-2}$$

とすべきである．電圧（アクロス変数）については

$$E_0 = E_n \tag{6-3}$$

$$E_1 = E_p \tag{6-4}$$

とすればよい．

RL 回路についても同様に，因果的端子と非因果的端子の信号間の関係は，図 6-9 のように

$$I_1 = -I_n \tag{6-5}$$

$$I_2 = -I_p \tag{6-6}$$

$$E_1 = E_n \tag{6-7}$$

$$E_2 = E_p \tag{6-8}$$

となる．

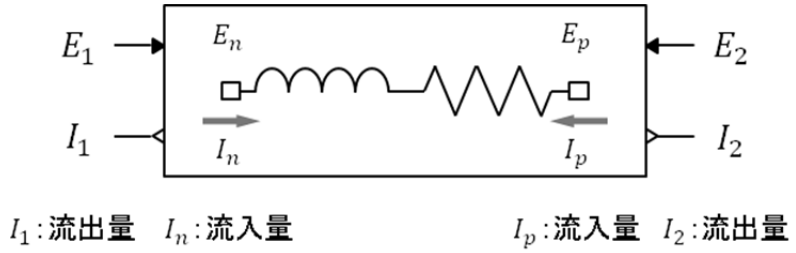


図 6-9　RL モデルの各端子の極性

このように，因果的端子と非因果的端子の間の信号を，極性を考慮して等式で接続すれば，非因果的モデルを FMU 化することが可能となる．

しかし，実際に非因果的端子を持つモデルを FMU 化する場合においては，これらの関係式を毎回記述していては，符号の誤りが生じる可能性が高い．そこで，本ガイドラインでは以下で定義する，非因果的端子と因果的端子の間の接続式をあらかじめ記述した「アダプタ」の使用を推奨する．例えば，上記の電圧源モデルの両側の非因果的端子のそれぞれにアダプタを付加すると図 6-10 のように，数式を記述することなく因果的端子と接続することが可能となる．

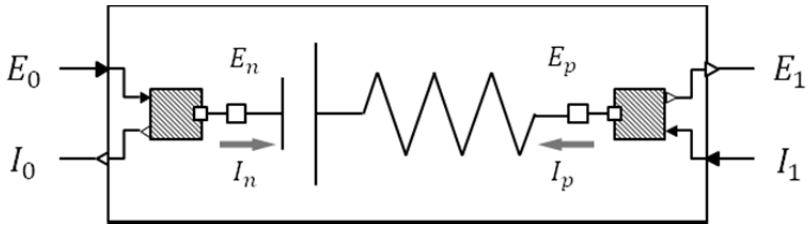


図 6-10 アダプタを用いた電圧源モデルの因果的接続

この時，アダプタ内部の数式がどのようになっているかを考える．なお，電圧（アクロス変数）については 6.2.1.2 で述べたように全て極性の反転は起こり得ないため，説明を割

愛する．図 6-11 のように，電圧源モデルの－端子に付加したアダプタの非因果的端子の電流値（I_n'）はキルヒホッフの電流則から，

$$I_n' + I_n = 0 \Rightarrow I_n' = -I_n \tag{6-9}$$

一方，因果的端子との接続においては，

$$I_0' = I_0 \tag{6-10}$$

となる．

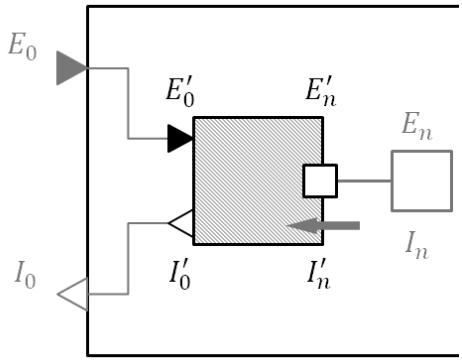


図 6-11 電圧源モデルの－端子のアダプタによる接続

モデルの物理量（I_n，E_n）と因果的端子の信号（I_0，E_0）の間では，(6-1)式の通り電流の極性が反転しなければならないため，アダプタ内部では

$$I_0' = I_n' \tag{6-11}$$

と，反転のない関係となる．このアダプタの因果的端子は電流を出力するものであり（すなわち「流出量」を意味する），モデル内部の非因果的端子の電流は「流入量」を意味することから，一見，極性が反転すべきだと予想されるが，モデルの非因果的端子とアダプタの非因果的端子の接続によって，I_nとI_n'の間で既に反転が生じているため，このような関係式となる．

また，電圧源モデルの＋端子についても，

$$I_p' + I_p = 0 \Rightarrow I_p' = -I_p \tag{6-12}$$

$$I_1' = I_1 \tag{6-13}$$

となり，(6-2)式からアダプタ内部では

$$I_1' = -I_p' \tag{6-14}$$

となり，極性を反転させなければならない．

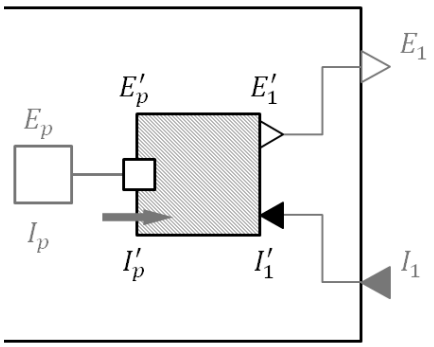


図 6-12 電圧源モデルの＋端子のアダプタによる接続

つまり，アダプタのモデル式は以下の通りとなる.
スルー変数（電流）出力アダプタ

$$E_{acausal} = E_{causal} \tag{6-15}$$

$$I_{acausal} = I_{causal} \tag{6-16}$$

アクロス変数（電圧）出力アダプタ

$$E_{acausal} = E_{causal} \tag{6-17}$$

$$I_{acausal} = -I_{causal} \tag{6-18}$$

「電流・電圧」の組み合わせを持つ電気系の端子に限らず，このアダプタの考え方は「スルー変数・アクロス変数」の組み合わせを持つ全てのドメインについても適用可能である.

6.3 モデル接続の考え方

6.2 節では，非因果的モデルをアダプタにより，因果的モデルに変換して FMU を作成する方法について述べた．本節では逆に，FMU を読込み接続するときの考え方について説明する．想定されるケースとしては，様々な部署や企業から持ち寄られた FMU を相互接続する場合が考えられる.

一般に，他部署・他企業から受け取った FMU を，モデルにそのまま結合できるとは限らない．本節では，特に検討する必要があると考えられる２つの接続形態について説明する.

6.3.1 FMU の接続先が非因果的モデルの場合

接続先のモデルが非因果的モデルだった場合，FMU の端子が因果的端子なのだから接続はできない．この場合，因果的端子を非因果的端子に変換するアダプタが必要となる．図 6-13 は，因果的モデル（FMU）の電圧源をアダプタ（太い破線で囲った部分）によって非因果的モデルに接続する様子を表している.

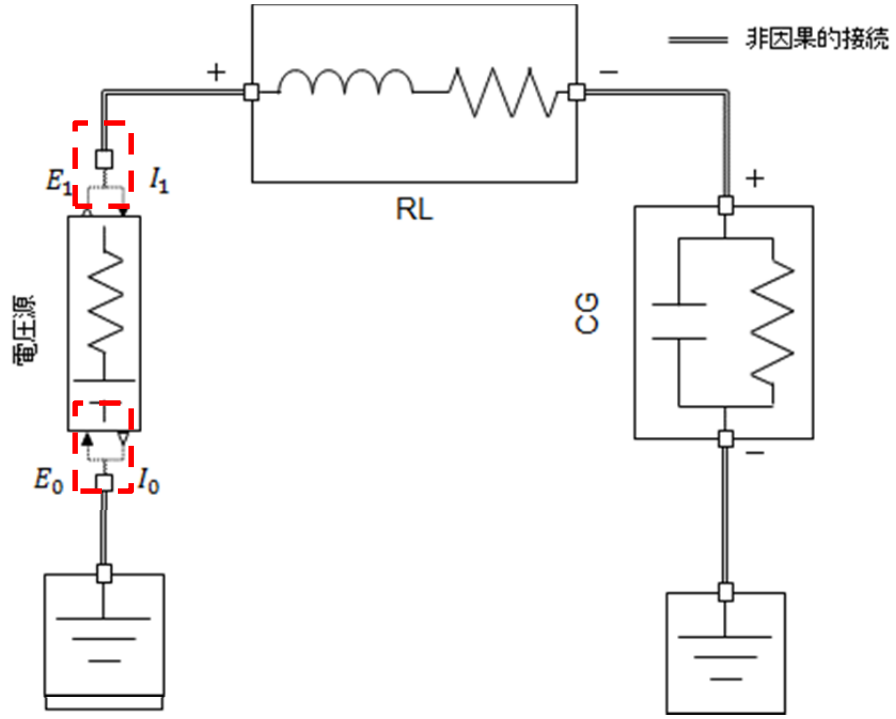


図 6-13　因果的モデル（FMU）の非因果的接続のイメージ

電圧源モデルの二つの端子について考えると，図 6-14 のように電流を出力する－端子の電流値 I_0 は流出量を意味するのに対し，それに接続するアダプタの非因果的端子の電流 I_0' は Modelica Standard Library のルールから，流入量を意味する．そのため，このアダプタでは電流の極性反転が必要である．一方，＋端子ではどちらも流入量を意味するため，極性の反転は不要である．つまり，

スルー変数（電流）出力アダプタ（図 6-14 の＋端子側のアダプタ）

$$E_{acausal} = E_{causal} \tag{6-19}$$

$$I_{acausal} = I_{causal} \tag{6-20}$$

アクロス変数（電圧）出力アダプタ（図 6-14 の－端子側のアダプタ）

$$E_{acausal} = E_{causal} \tag{6-21}$$

$$I_{acausal} = -I_{causal} \tag{6-22}$$

となる．これらの関係式は(6-15)式～(6-18)式と同じである．そのため，FMU（あるいは因果的モデル）を非因果的接続する際に使用するアダプタは，非因果的モデルに因果的端子を付加するために使用するアダプタと共通のものを使用することができ，これらのアダプタは「スルー変数（電流）を出力するアダプタか，アクロス変数（電圧）を出力するアダプタか」だけを区別して使用すればよい．（図 6-15）

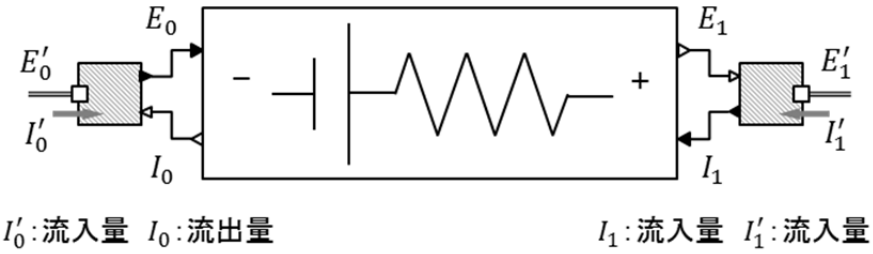


図 6-14　電圧源の因果的モデルのアダプタによる非因果的接続

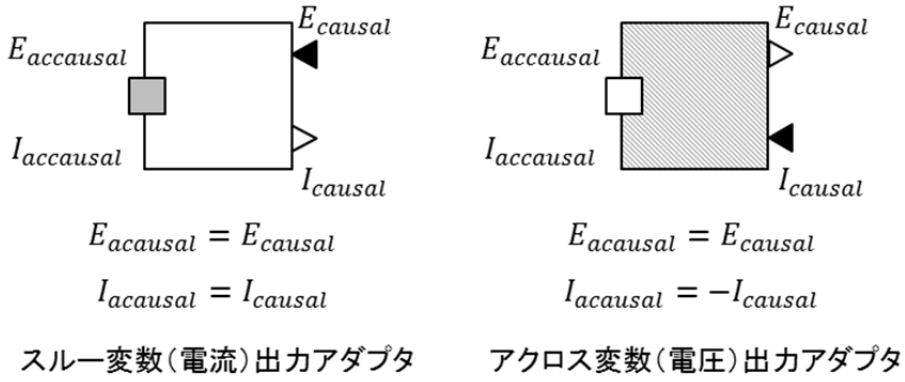


図 6-15　因果的端子，非因果的端子の接続に使用するアダプタ

6.3.2　FMU の出力と，接続先のモデルの入力が異なっている場合

次に，FMU の出力変数（または入力変数）と接続先の入力変数（または出力変数）が異なっている場合について考える.

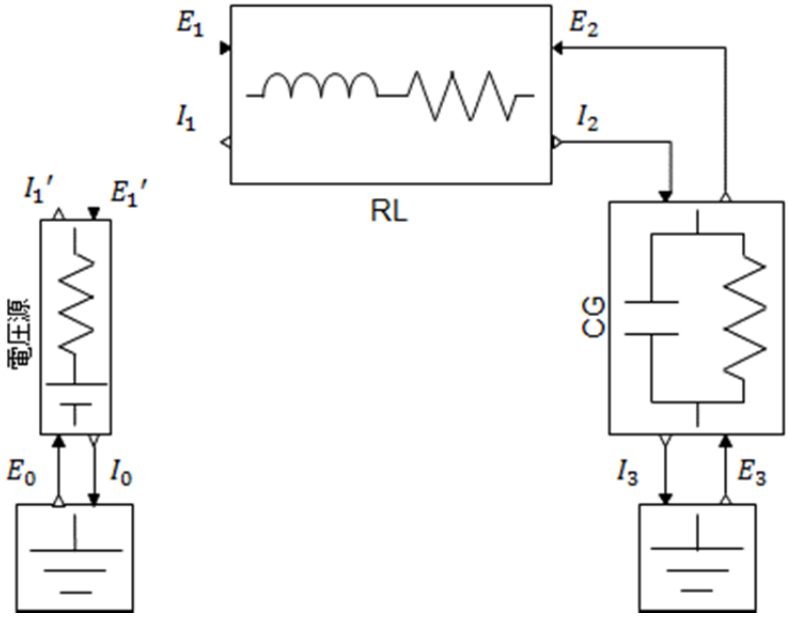


図 6-16　同じ変数を出力する FMU 同士をつなげる場合

図 6-16 では，電圧源と RL 回路がともに電流を出力し，電圧を入力しているためこれらを直接結合することができない.

こうした FMU を結合するためには, いったん因果的端子を非因果的端子にそれぞれ変換し，その非因果的端子同士を結合すればよい（図 6-17 の斜線領域）. これにより，信号の受け渡しはツールが処理してくれる.

なお，前節と同様の理由で，ここでのアダプタにも図 6-15 に示したものをそのまま使用することができる.

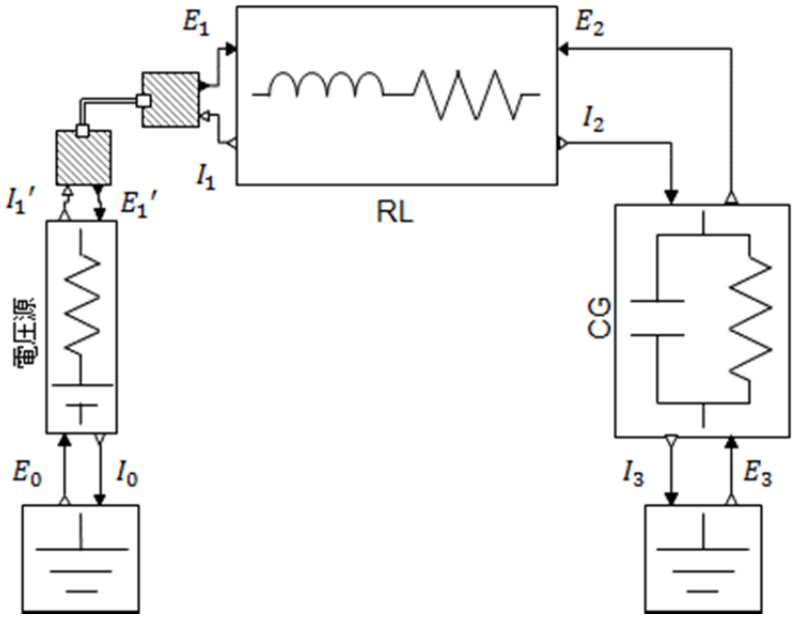


図 6-17　アダプタを使って因果的端子を非因果的端子に変換して接続する様子

ただしこの方法は，ツールが非因果的モデリングに対応していることを前提としている．因果的モデリングを前提としているツールではこの方法が使えない．

6.4　JSAE 推奨アダプタ

6.2 節と 6.3 節では，モデルを分割する際と結合する際に同じアダプタが必要になることを，電気回路の例を用いて説明した．こうしたアダプタを 6.2 節で説明したルールに従ってあらかじめ作成しておき，状況に応じて参照できるようにしておくことは有意義であろう．

本 WG では，6.2.1 項で定義した極性ルールに基づいて，Modelica 協会から提案のあった変換用のアダプタを検証し，一部の極性変更を Modelica 協会へ提案した．以下の項で，それらを紹介する．

なお，ここで紹介するアダプタはすべて，Modelica 言語を用いて実装しているため，他の言語に基づくツールでは使用できない．その場合，同様のアダプタを，ツールが用いている言語に基づいて作成すればよい．

6.4.1 電気系アダプタ

- 電流信号出力アダプタ “PinToI”

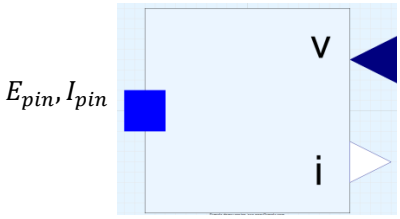


図 6-18 電流信号出力アダプタ “PinToI”

非因果的端子から，電流を出力としてとりだすアダプタであり，電流（スルー変数）の符号を反転させない．図中の記号を使えば，

$$E_{pin} = v \tag{6-23}$$

$$I_{pin} = i \tag{6-24}$$

となる．

- 電圧信号出力アダプタ “JsaePinToV”

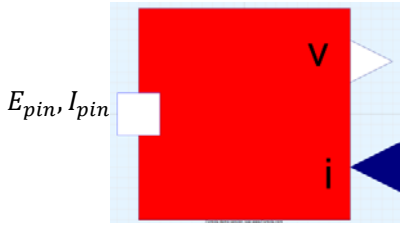


図 6-19 電圧信号出力アダプタ “JsaePinToV”

非因果的端子から，電圧を出力としてとりだすアダプタで，電流（スルー変数）の符号を反転させる．図中の記号を使えば，

$$E_{pin} = v \tag{6-25}$$

$$I_{pin} = -i \tag{6-26}$$

となる．

6.4.2 回転系アダプタ

- トルク信号出力アダプタ “JsaeRotFlangeToTau”

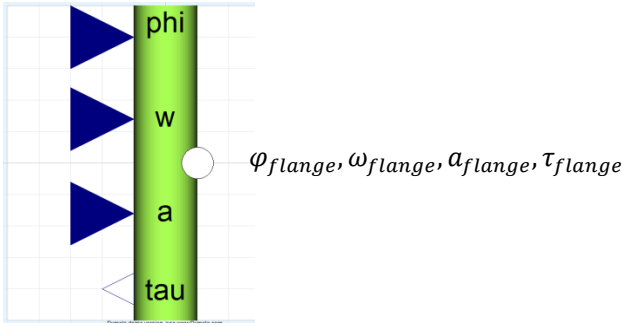


図 6-20　トルク信号出力アダプタ“JsaeRotFlangeToTau”

非因果的端子から，トルクを出力としてとりだすアダプタである．電気系のアダプタと同様の考察から，トルク（スルー変数）の極性を反転させない．図中の記号を使えば，

$$\varphi_{flange} = phi \tag{6-27}$$
$$\omega_{flange} = w \tag{6-28}$$
$$a_{flange} = a \tag{6-29}$$
$$\tau_{flange} = tau \tag{6-30}$$

となる．

- 回転角信号出力アダプタ“JsaeRotFlangeToPhi”

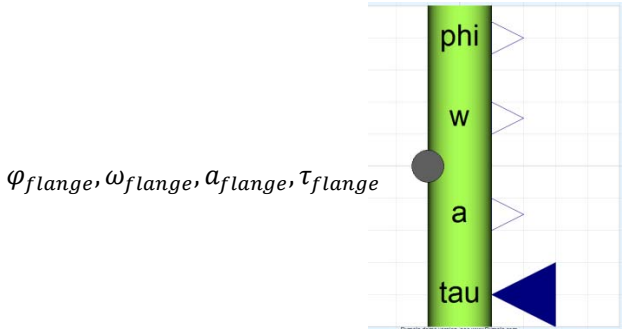


図 6-21　回転角信号出力アダプタ“JsaeRotFlangeToPhi”

非因果的端子から，角度を出力としてとりだすアダプタである．電気系のアダプタと同様の考察から，トルク（スルー変数）の極性を反転させる．図中の記号を使えば，

$$\varphi_{flange} = phi \tag{6-31}$$
$$\omega_{flange} = w \tag{6-32}$$
$$a_{flange} = a \tag{6-33}$$
$$\tau_{flange} = -tau \tag{6-34}$$

となる．

6.4.3　並進系アダプタ

- 力信号出力アダプタ“JsaeTransFlangeToF”

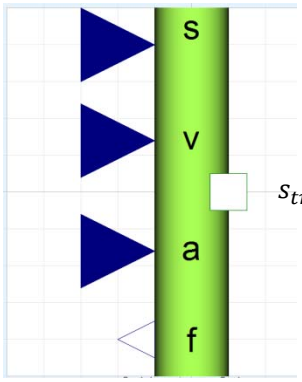


図 6-22　力信号出力アダプタ “JsaeTransFlangeToF”

非因果的端子から，力を出力としてとりだすアダプタである．電気系のアダプタと同様の考察から，力（スルー変数）の極性を反転させない．図中の記号を使えば，

$$s_{trfln} = s \tag{6-35}$$
$$v_{trfln} = v \tag{6-36}$$
$$a_{trfln} = a \tag{6-37}$$
$$f_{trfln} = f \tag{6-38}$$

となる．

- 位置信号出力アダプタ “JsaeTransFlangeToS”

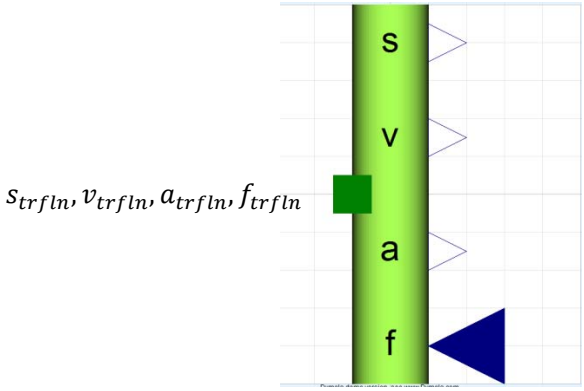


図 6-23　位置信号出力アダプタ “JsaeTransFlangeToS”

非因果的端子から，位置を出力としてとりだすアダプタである．電気系のアダプタと同様の考察から，力（スルー変数）の極性を反転させる．図中の記号を使えば，

$$s_{trfln} = s \tag{6-39}$$
$$v_{trfln} = v \tag{6-40}$$
$$a_{trfln} = a \tag{6-41}$$
$$f_{trfln} = -f \tag{6-42}$$

となる．

以上が，（“PinToI”を除き）自動車技術会から提案したアダプタの定義である．現時点では電気系，機械系回転，機械系並進のドメインでのアダプタを定義しているが，今後熱系や流体系などのドメインでのアダプタの定義もしていく必要がある．

6.5 FMU の CS と ME について

5.2.1 項で説明したように，FMU には大きく分けて，Model Exchange（ME）と Co-Simulation（CS）の２種類がある．その定義はすでに述べたが，本節では，それらを実際に用いる際の性質の違いやメリットとデメリットについて説明する．

6.5.1 ME の特徴

5.2.1 項で述べたように，ME は，インポートした FMU を含むモデル全体で同一のソルバを用いる方式であり，FMU 内部にはソルバが含まれていない．したがって，ME の FMU を使用するには，使用するシミュレーション環境でソルバが提供されることが前提となる．

シミュレーション環境で代数ループソルバが利用可能な場合，モデルの初期状態を計算する際の代数方程式を，反復計算等を用いて解くことができる．ただし，FMU を作成した環境のソルバと FMU を取込んだ環境のソルバが異なるとシミュレーション結果に差が発生する可能性がある．したがって，作成側での FMU 単体特性と取込み側での FMU 単体特性の比較検証を行った上で FMU 接続することが必要となる．

6.5.2 CS の特徴

CS は ME とは異なり，FMU 自身がソルバを持っており，FMU 内部のモデル計算にはそのソルバが用いられる．したがって，ソルバを持たないツールであっても計算を行うことができる．

FMU のモデルは内部のソルバを使用して計算するので，一般に，FMU 内の時間刻みと外部の時間刻みとは一致しない．外部との信号のやり取りは通信間隔ごとに行われる．この通信間隔は外部の時間刻みに等しいか，長くなければならないため，外部のモデルが FMU のモデルから信号を受け取るのもそれだけ遅れることになる．

通信間隔の合間は，マスタ/スレーブ間の入出力は，前回の通信時間での結果を一定値として取り扱う，もしくは FMI2.0 にてやり取りができるようになったヤコビ行列や偏微分係数を使用して予測する必要がある．前者はシミュレーションツールの現状のケースにほぼあてはまり，その場合はマスタ側が代数ループソルバを持っていたとしても，FMU を含めた代数ループがある場合は解くことができない．後者の場合は，FMU を含めた代数ループを解く手法は存在するが，現状対応しているツールはなく，将来の機能拡張が期待される．

6.5.3 ME と CS の違い

以上が ME と CS の特徴である．これらを表にまとめると，表 6-3 のようになる．モデル作成者は，これらの特徴を踏まえたうえで，ME か CS かを選択する必要がある．

表 6-3　ME と CS の特徴比較表

	ME	CS
ソルバの選択の自由度	・モデル全体で選択可 ・すべての FMU のモデルを同じソルバで動かさなくてはならない	・FMU のモデルごとにソルバを変えられる ・FMU を作成した時の設定に縛られる
ソルバを持っていないツールでの使用	不可	可
FMU の入力データ更新タイミング	時間刻みごと	通信間隔ごと（時間刻みよりも長い）

前項で述べたとおり，CS は内部にオリジナルのソルバを持っているので，CS FMU を接続することで代数ループが形成される場合は，異なる結果となる場合もある．そこで簡単なモデルでこれを示す（図 6-24）．

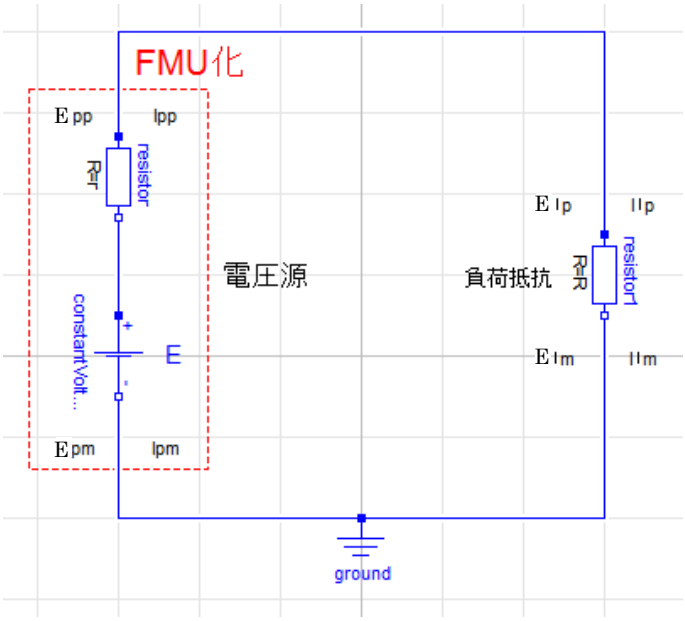


図 6-24　簡単な回路モデル

この回路は電圧源と負荷抵抗をつないだもので，電圧源は内部抵抗を持つ．この電圧源を，内部抵抗を含めて FMU 化する．FMU 化する際は，先に導入したアダプタを用いて図 6-25 のように入出力を定義する．

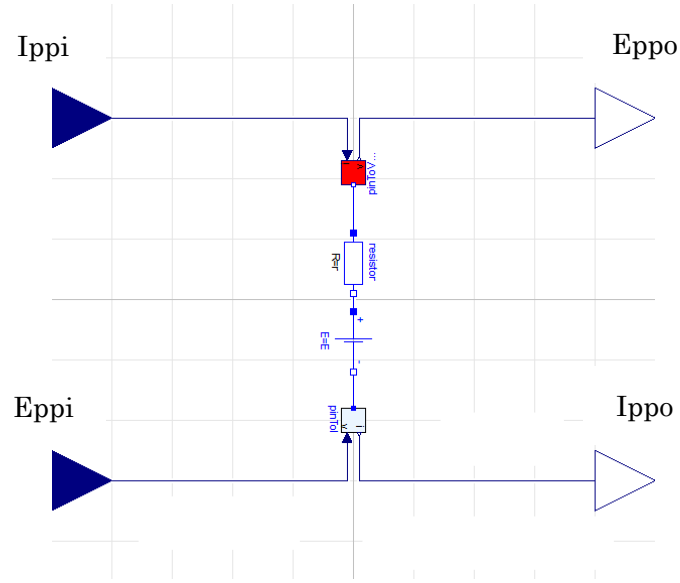


図 6-25　電圧源の FMU 化

FMU 化した電圧源を，アダプタを用いて負荷抵抗に再び接続する（図 6-26）．この図では ME を接続しているが，CS も同様である．

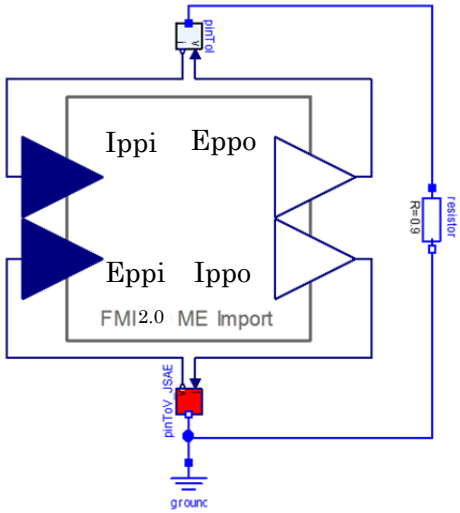


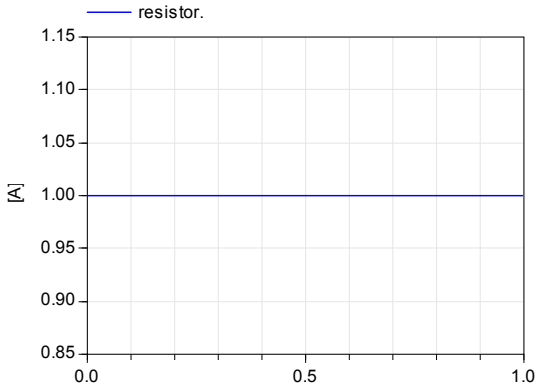
図 6-26　FMU 化した電圧源を再び負荷抵抗に接続する様子

図 6-26 を見ると，電圧源の FMU も，抵抗とアダプタを合わせた部分も直接フィードスルーブロックであり，それらがループをなしているため，このモデルは代数ループとなっている．

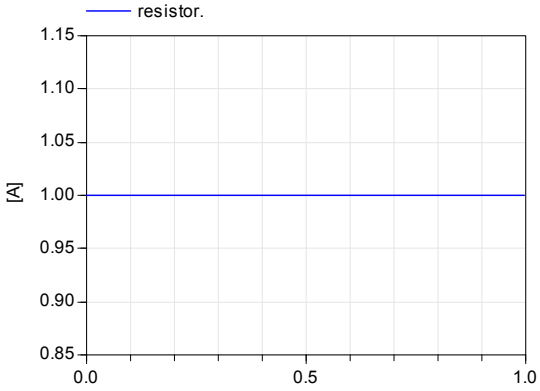
図 6-27 は，負荷抵抗を流れる電流値を計算した結果である．4 つのプロットのうち上段左側は，オリジナルのモデルを作成したツールを用いて計算したもの，右側は ME で FMU 化した場合の結果である．ME で計算した場合，オリジナルのモデルを再現できている．代

数ループがあるにもかかわらず，正しい結果が得られたのは，使用したツールの代数ループソルバが働いたためである．

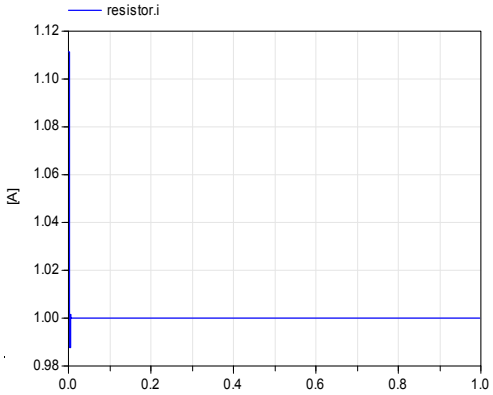
これに対し，図 6-27 の下段左にある CS の結果を見ると，初期時刻付近に値の振れがあることがわかる．このプロットの初期時刻付近を拡大したのが下段右側の図であり，振動しながら本来の値に収束していることがわかる．



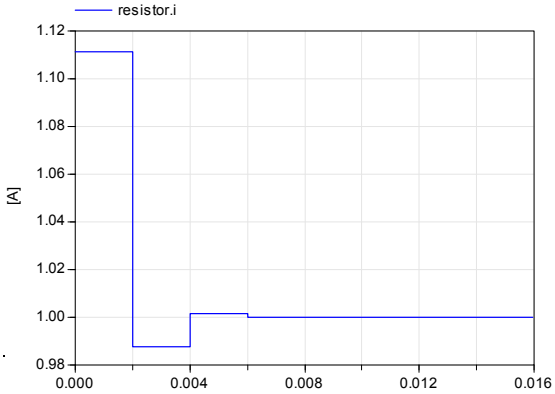
(a) オリジナルモデル



(b) ME FMU 使用



(c) CS FMU 使用



(d) CS FMU 使用（0 秒近傍拡大図）

図 6-27　シミュレーション結果

CS の結果の振動は，通信間隔ごとに電圧源と負荷抵抗の反復計算を行った結果で，本来の動作とは異なるものである．

ただし，今回示したケースでは計算結果が収束したが，パラメータの値によっては一巡伝達関数のゲインが 1 を超えてしまい，結果が発散する可能性がある．こうした理由から，モデルが代数ループを含んでいる場合には，ME と CS のどちらが適切かをツールごとに判断したうえで選択することが必要である．

6.6 FMU を含むシステムモデル例

ここでは本 WG が FMU の作成と読込みの実行確認をするために使用したベンチマークモデルから２つの簡易システムモデル例を紹介する.

なお，モデル接続に使用した FMU は ME である.

6.6.1 電源系の簡易電気回路システム例

ベンチマークモデルの１つ目を図 6-28 に示す．この回路は電圧源，電流源，負荷抵抗などが並列接続された電気回路で，電圧源をバッテリ，電流源を発電機の発電電流などに当てはめると自動車の電源系回路と考えることもできる.

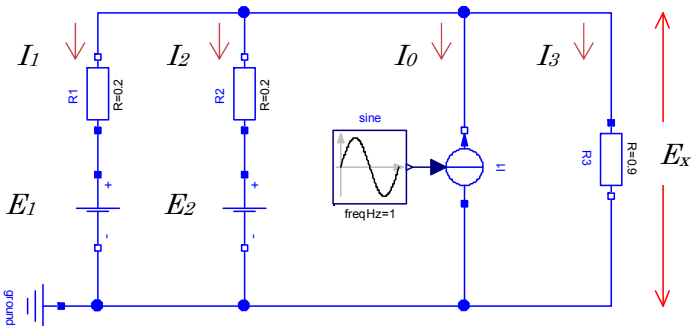


図 6-28 簡易電気回路システム

この簡易電気回路システムには以下に示す関係式が成り立つ.

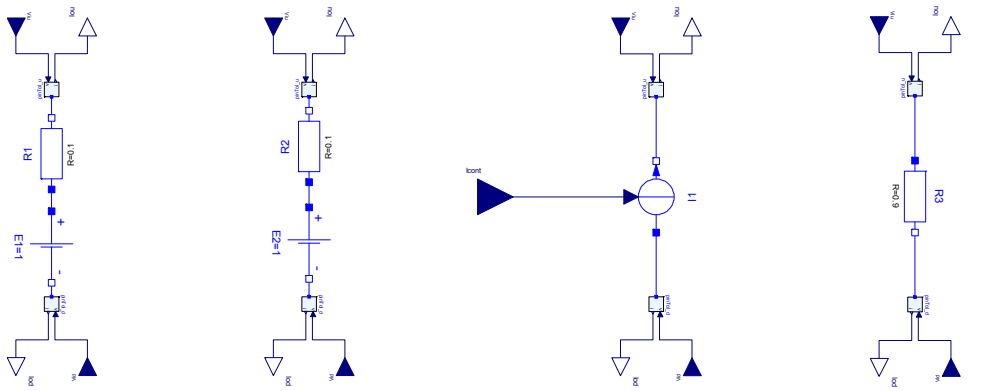
$$I_0 + I_1 + I_2 + I_3 = 0 \tag{6-43}$$

$$E_x = I_1 \cdot R_1 + E_1 \tag{6-44}$$

$$E_x = I_2 \cdot R_2 + E_2 \tag{6-45}$$

$$E_x = I_3 \cdot R_3 \tag{6-46}$$

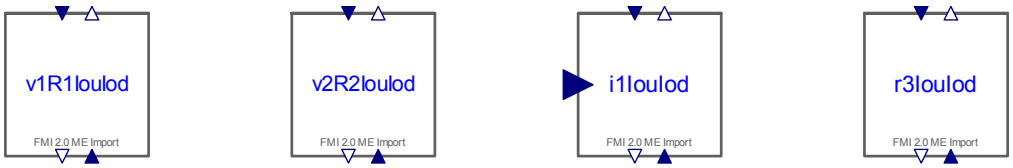
ここで並列接続される各電気部品を FMU 化することを考える．この際，6.4.1 項で述べた電気系アダプタを使用し，図 6-29 の様にモデル分割を行う.



電圧源１　　電圧源２　　電流源　　負荷抵抗

図 6-29　簡易電気回路システム FMU 化のための分割モデル

その後，各分割モデルから FMU を作成する．作成された FMU をツールで読込むと，図 6-30 のような FMU モデルが確認できる．



電圧源１　　電圧源２　　電流源　　負荷抵抗

図 6-30　結合するツールに読込まれた簡易電源回路システム構成要素の FMU モデル

次は，読込んだ FMU モデルの接続を行う．モデルの接続設計を行う担当者は，各 FMU モデルの入出力信号とモデル結合後のシステム構成（今回の例では図 6-28 のシステム構成）を十分把握しておくことが必要である．システム構成の把握を確実にするためには，FMU モデルを入手する前に，FMU の入出力を規定した中身のない空のブロックなどを用いてシステム構成の設計を完了させておき，FMU モデルが入手できたら置き換えるというような対応が有効と考えられる．

FMU モデルの接続例として，読込んだ FMU モデルに 6.4.1 項で述べたアダプタを接続し，各 FMU モデル間を非因果的に接続したものを図 6-31 に示す．

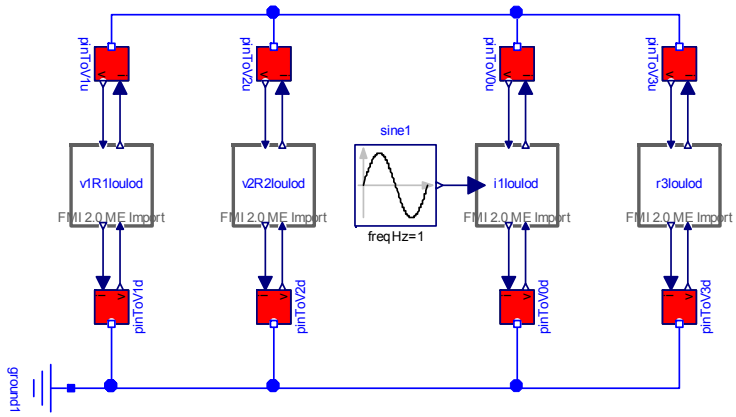


図 6-31　FMU モデルにアダプタを接続した非因果的モデル接続

また，読込んだ FMU モデルをそのまま因果的に接続した例を図 6-32 に示す．

因果的に FMU モデルを接続するためには(6-43)式～(6-46)式の関係をモデル記述として追加する必要がある．このシステム例では各 FMU モデルは電圧信号を入力として電流信号を出力する．そこで(6-43)式の関係を満たすために全ての電流の合計を求め，その合計値が目標値である 0 [A]になるような E_x を算出し，各 FMU モデルの入力信号として接続している（図 6-32 の右上部分）．

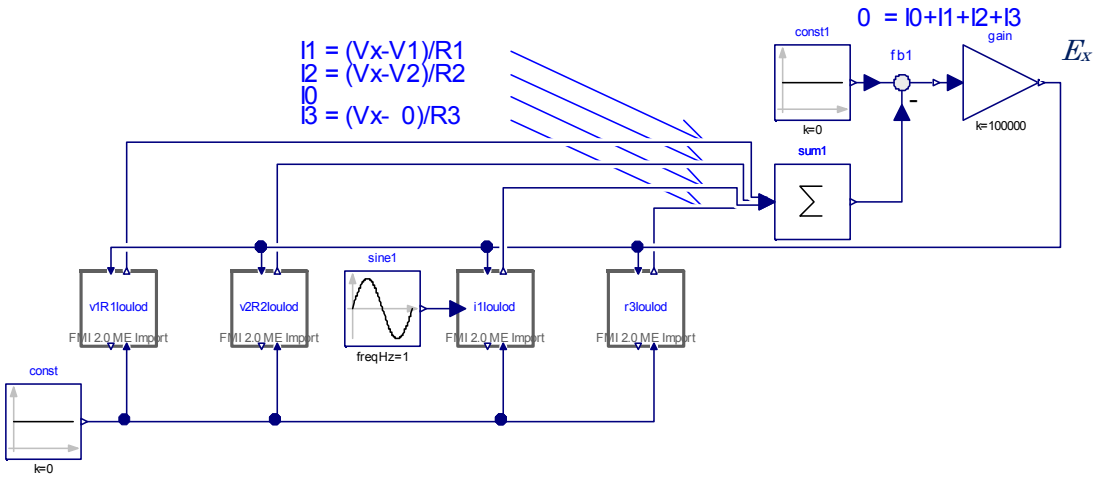


図 6-32　因果的に FMU モデルを接続した例

以上の様に，FMU モデルを用いて電気回路の並列接続を行う場合には非因果的な接続のほうが，モデル接続のための追加記述設計が少なくて済む．

接続後のシミュレーション結果を図 6-33 に示す．非因果的接続，因果的接続共に分割前のシステムモデルのシミュレーション結果と一致していることが分かる．

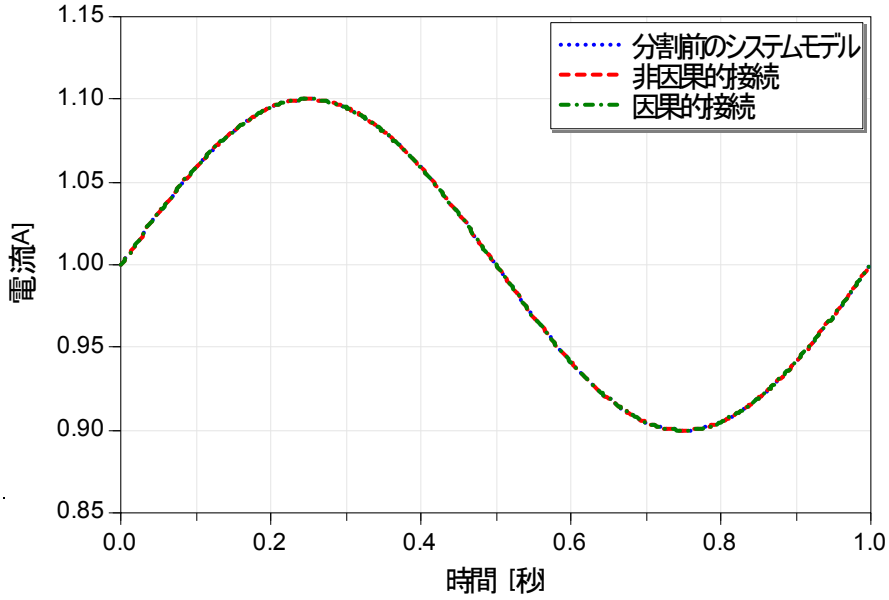


図 6-33　簡易電気回路システムのシミュレーション結果（R_3 電流）

6.6.2　モータを用いた簡易制御システム例

ベンチマークモデルの 2 つ目を図 6-34 に示す．この制御システムは図 6-35 に示すようなモータの回転角制御システムで，構成要素は制御コントローラ，DC モータ，回転運動系の 3 つのサブシステムから構成される．

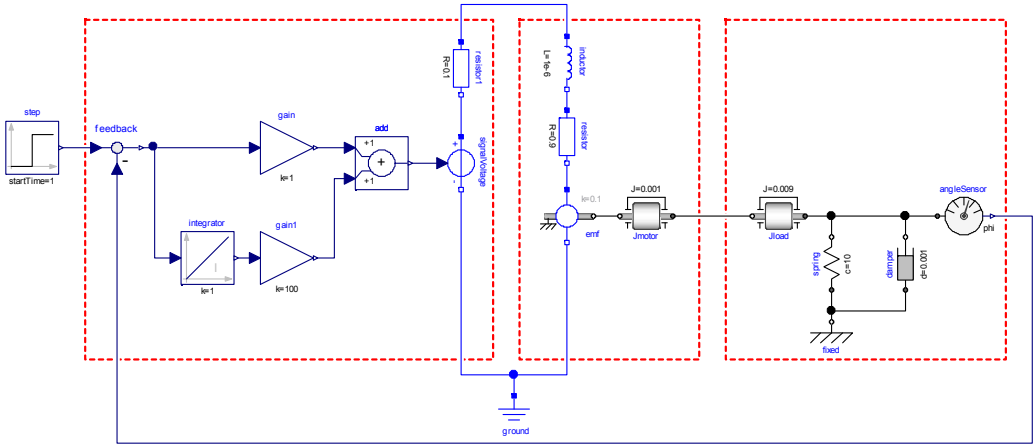


図 6-34　簡易制御システム

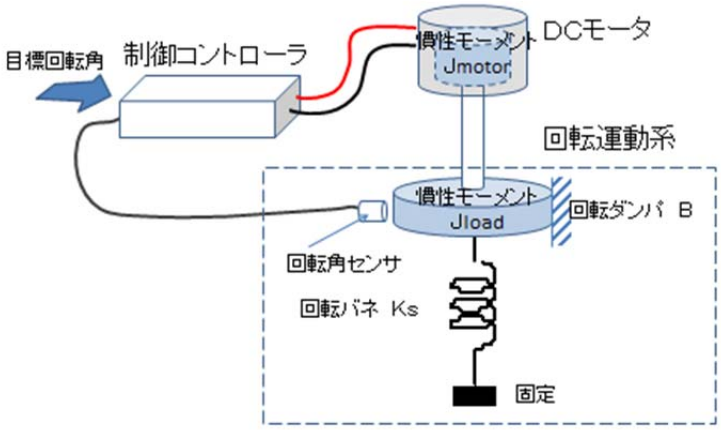


図 6-35　モータ回転角制御システムイメージ図

前の例と同様に 3 つの構成要素を FMU 化することを考える．この際，6.4.1 項の電気アダプタに加えて 6.4.2 項の回転系アダプタも使用し，図 6-36 の様にモデル分割を行う．

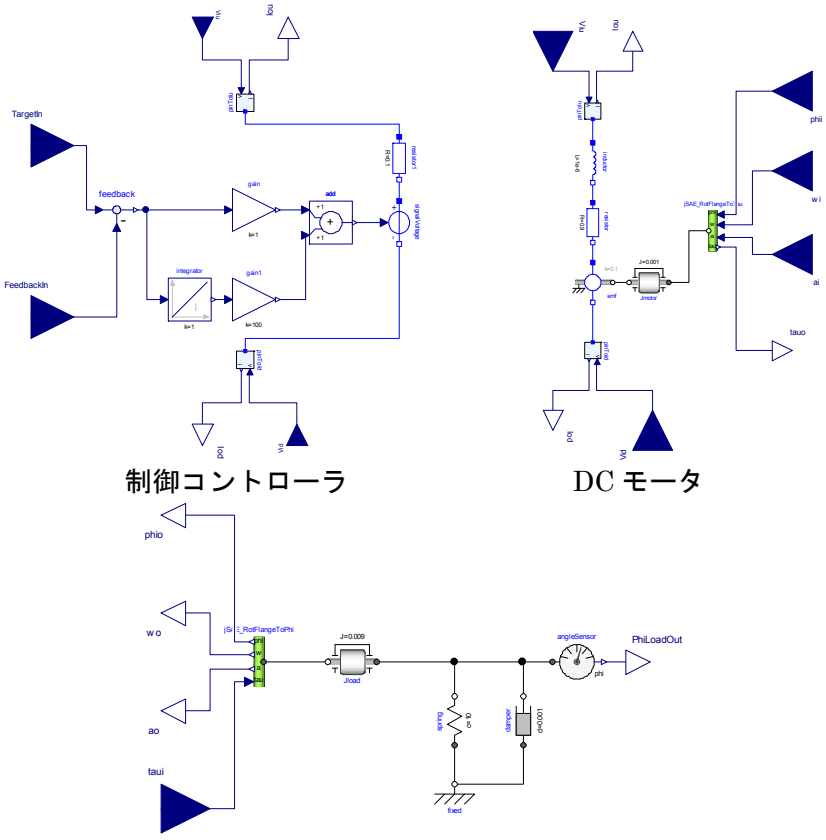


図 6-36　簡易制御システム FMU 化のための分割モデル

各分割モデルから FMU を作成し，読込側ツールに読込むと，図 6-37 のような FMU モデルが確認できる．

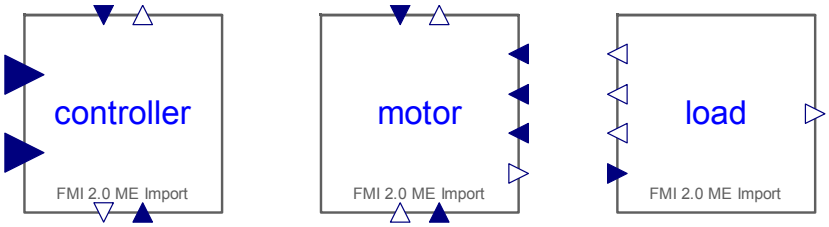


図 6-37　読込側ツールに読込まれた簡易制御系システム構成要素の FMU モデル

読込んだ FMU モデルを因果的に接続した例を図 6-38 に示す．コントローラ側の電流とモータ側の電流の合計が 0 [A]になる様にモータへの印加電圧を算出しているのは前の例と同様である．DC モータと回転運動系の間では，DC モータと回転運動系の回転軸の結合を回転角，回転角速度，回転角加速度及びトルクの信号という形で接続している．

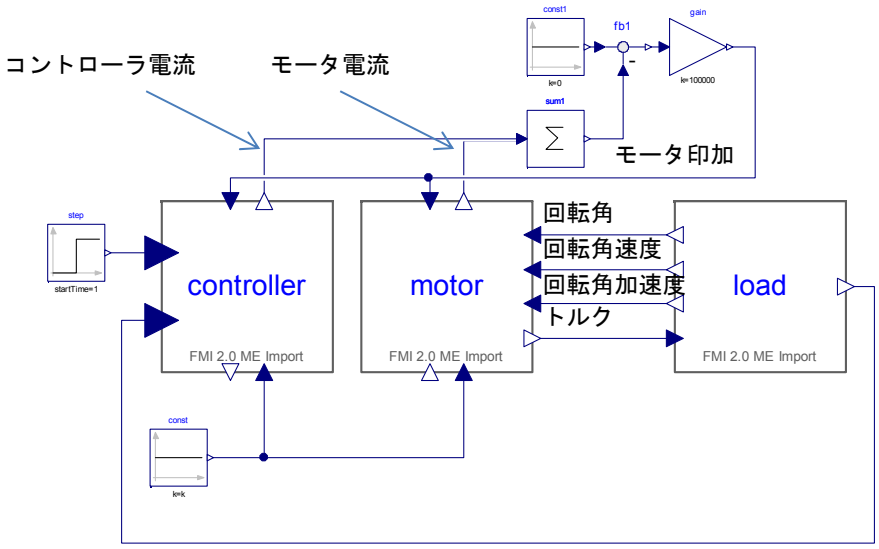


図 6-38　因果的に FMU モデルを接続した簡易制御システムモデル例

図 6-37 に示した FMU モデルでは，回転軸での結合信号の入出力の向きが DC モータと回転運動系で矛盾なく接続できる状態であった．しかし，必ずしも相互の信号の向きが一致するとは限らない. 例えば回転運動系の FMU モデルが図 6-39 に示す様にトルクを出力し，回転角，回転角速度，回転角加速度を入力としている場合，このままではモータ回転軸との結合ができない．このような状況を回避するために非因果的な接続を適用した接続例を図 6-40 に示す．

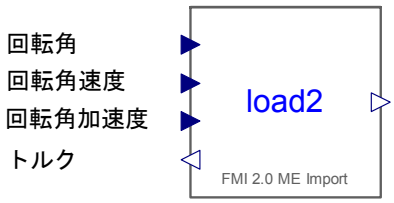


図 6-39　トルク出力，回転角，回転角速度，回転角加速度入力の FMU モデル

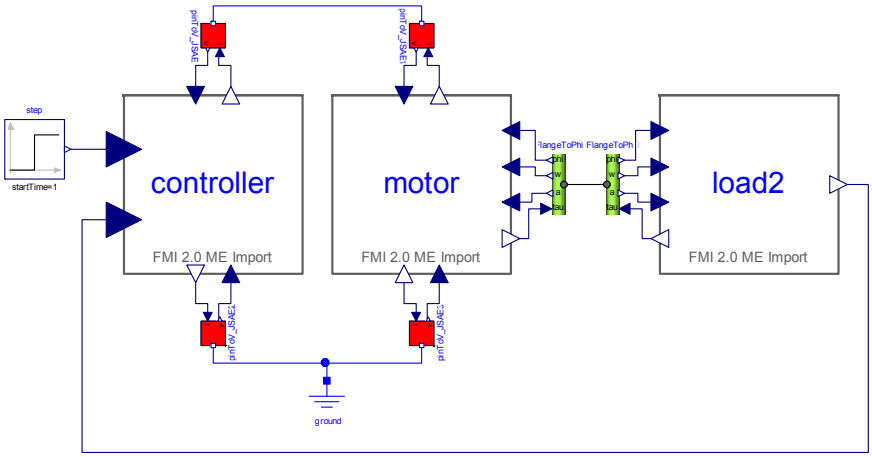


図 6-40　非因果的に FMU モデルを接続した簡易制御システムモデル例

この様に，非因果的な接続は FMU モデルの接続自由度を高めることができる．

接続後のシミュレーション結果を図 6-41 に示す．非因果的接続，因果的接続共に分割前のシステムモデルのシミュレーション結果と一致していることが分かる．

なお，モデルのパラメータは，シミュレーション結果比較のためにあえて振動状態となる設定としている．

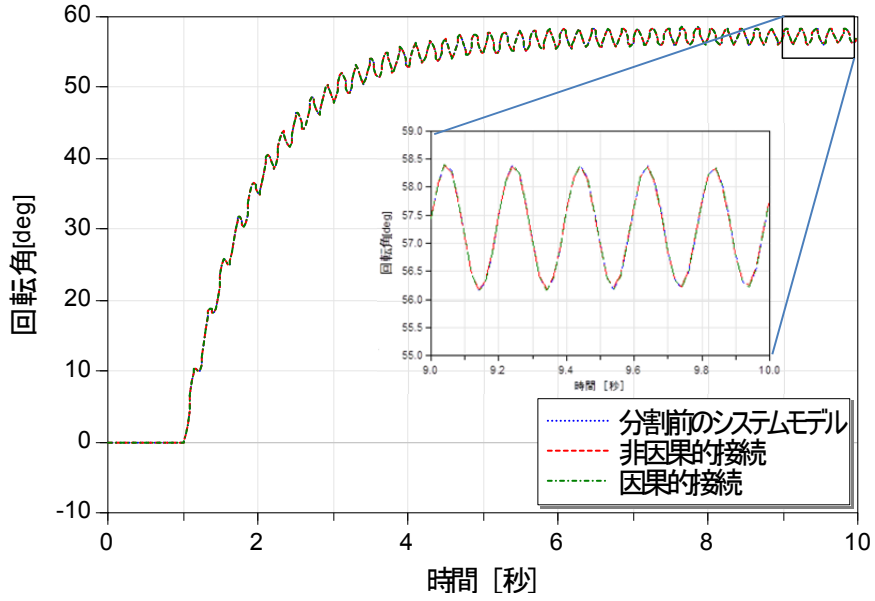


図 6-41　簡易制御システムのシミュレーション結果（負荷軸回転角）

6.7　FMU を含むシステムモデルの計算実行の注意点

6.7.1　代数ループ（Algebraic loop)問題への対応

3.4 節に述べたように，代数ループが存在すると，FMU モデルを含むシミュレーション実行に大きな問題が生じる．一つの FMU 提供元のモデルに代数ループが含まれている場合，代数ループが生じないように元のモデルの定式化を見直すか，やむを得ず遅延要素を入れて擬似的に解いている場合，3.4 節で述べたように，代数ループの一巡伝達関数のゲインの絶対値が１未満であることを確認する必要がある．

一般に，代数ループは，電気系で抵抗要素を並列接続したり，機械系で，リジッドなギヤ要素を直列接続したりするなど，要素単独の時は，スルー変数が独立だが，接続により，スルー変数が，微分演算関係を含まない代数的な関係で拘束を受ける場合に発生する．一つの FMU 提供元モデルの中で，このような接続関係が生じる場合，非因果的モデリングツールで，代数ループが解ける形で FMU オブジェクトファイルを作成するか，代数ループのない数式処理が可能な形でモデル式を記述する必要がある．

一つの FMU 提供元のモデルでは代数ループが含まれていなくても，複数の FMU を組み合わせた時に代数ループができる可能性もある．この場合は，非常に厄介である．基本的には，大きな代数ループが生じないように，各 FMU 提供元のモデルの定式化を変える必要がある．それが困難な場合，代数ループの中の一部に，遅延要素を入れる方法はあるが，3.4 節で述べた様に，代数ループの一巡伝達関数のゲインの絶対値が１未満であることを確認する必要がある．さもないと，結果が振動したり，発散したりすることになる．遅延要素を入れて安定解に収束させることもできない場合，代数ループを生じさせている変数の

出力に，擬似的に，非常に小さな時定数の一次遅れ要素などの微分演算要素を挿入して，代数的拘束を解く方法もあるが，当然，この場合，解は近似的に解かれたものにしかならない．また，時定数の小さいダイナミクスを追加すると，系全体がスティフになりやすい．

6.7.2　スティフ系(Stiff　System)への対応

FMU 提供元のモデルがスティフな場合，CS，ME に関わらず，スティフ系に対応したソルバを採用することが望ましい．（CS の場合，FMU に埋め込むソルバもスティフ系に対応したものであることが望ましい．）複数の FMU を結合する場合，個々の FMU がスティフ系でなくても，それらの間の時定数が大きく異なる場合，結合後のモデルはスティフ系となる．この場合も，CS，ME に関わらず，スティフ系に対応したソルバを選択することが望ましい．但し，実装上の工夫が必要となるが，MultiRate Modeling で回避できる場合もある．

6.8　FMU 作成時のモデル記述注意点

現在の FMI の規格では，因果的なモデリングによって接続を行っている．このため，非因果的なモデリングの可能なモデリング言語とツールを用いる場合にも，因果的な接続を行うモデルを作成する必要がある．この方法のひとつとして，6.2.2 節で説明したアダプタを用いて因果的接続に対応する FMU を作成する方法を提示した．本ガイドライン作成にあたり，本 WG では，機械の回転系とアナログ回路を含む 4 つのモデルをベンチマークモデルとして，アダプタを用いた因果的モデリングについて調査を行った．その結果，特に電圧源と電流負荷および送電経路などを含む電気系回路モデルにおいて，微分を含むモデルの FMU を作成する際，初期値が定まらないために FMU の作成ができないという問題が生じることがわかった．本節では，電気系回路を例として，因果的接続に対応する FMU 作成する際に注意すべき点について述べる．

6.8.1　インダクタを含む回路

電気回路用アダプタには，因果的な接続をしようとするブロックの入出力が電圧入力・電流出力であるものと，電流入力・電圧出力であるものの 2 通りが存在する．図 6-42 に示す組み合わせで，インダクタと抵抗からなる単純な回路を例として，FMU を作成する場合を考える．

図 6-42 (a)は，直列につないだインダクタ L と抵抗 R_1 に，電流出力アダプタ(PinToI)と電圧出力アダプタ(PinToV)を接続している．図 6-42 (b)は，図 6-42 (a)に示した直列なインダクタ L と抵抗に対して，さらに並列に 1 MΩの抵抗 R_2 を接続している [13]．図 6-42 (c)では，2 つの電流出力アダプタをそれぞれ，直列につないだインダクタ L と抵抗 R_1 の両端に接続している．図 6-42 に示す 3 つの回路の内，(b)と(c)では FMU を作成可能であるが，(a)では FMU を作成することができない．このような回路での FMU の作成の可否は，h-パラメータと Y-パラメータ，2 つの 2 端子対パラメータを求めることで判断することがで

きる．なお，スルー変数に関して，本ガイドラインでは，入力端子では流入する方向を正，出力端子では流出する方向を正としている．しかしながら，6.8 節では 2 端子対パラメータの表記を慣例に合わせ，ブロック線図上での端子の入出力の設定によらず，数式中の電流の方向は，ブロックに流入する方向を正としている．このため，大きさが i である電流出力は，－i として取り扱われる．

図 6-42 (a), (b)はそれぞれ，(6-47)式と(6-48)式に示すとおり，h-パラメータで表される．また，図 6-42 (c)は(6-49)式のように Y-パラメータで表される．

$$\begin{bmatrix} e_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} z_{LR} & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ e_2 \end{bmatrix} \tag{6-47}$$

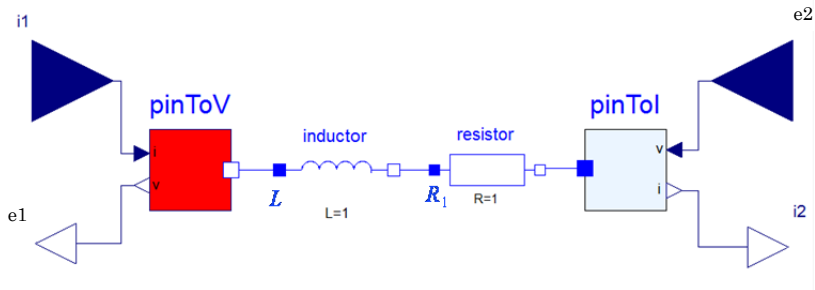
$$\begin{bmatrix} e_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} z_{LR\_R} & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ e_2 \end{bmatrix} \tag{6-48}$$

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \frac{1}{z_{LR}} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{6-49}$$

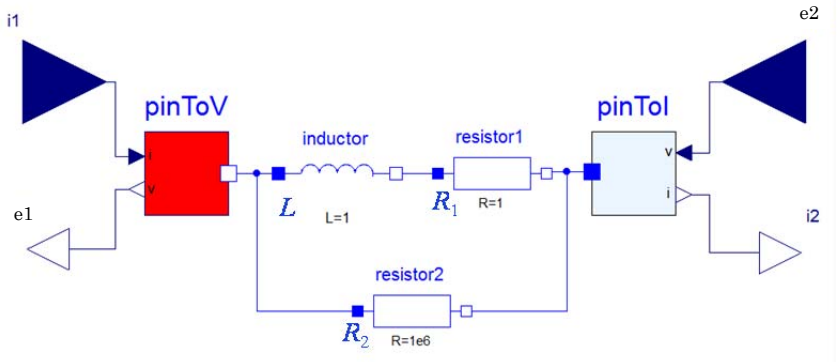
Z_{LR}および Z_{LR_R}は，それぞれの回路インピーダンスであり，各々以下の式のとおりである．

$$z_{LR} = sL + R_1 \tag{6-50}$$

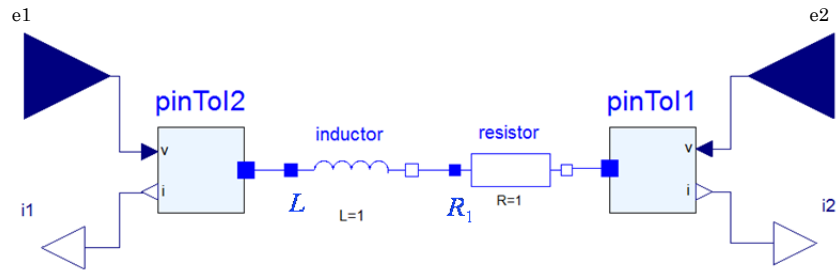
$$z_{LR\_R} = \frac{R_2}{1 + \dfrac{R_2}{sL + R_1}} \tag{6-51}$$



(a)　　LR 直列回路に電流出力アダプタと電圧出力アダプタを使用



(b)　　LR 直列回路と並列に抵抗を接続，電流出力アダプタと電圧出力アダプタを使用



(c)　　LR 直列回路の両端それぞれに電流出力アダプタを使用

図 6-42　FMU 作成に用いたインダクタと抵抗のモデル

ここで $s\to\infty$の極限を考えるとき，$1/Z_{LR}$および Z_{LR_R}は収束するが，Z_{LR}は発散する．これより，周波数領域が高いとき，図 6-42 (a)の回路では発散して状態が定まらず，図 6-42 (b), (c)の回路では収束して状態が定まることがわかる．

以上より，FMU の作成が可能な図 6-42 (b), (c)の回路では，2 端子対パラメータの要素が $s\to\infty$の極限，すなわち周期が無限小のとき収束することから，初期値が定まることがわかる．一方，図 6-42 (a)の回路では，2 端子対パラメータの要素が $s\to\infty$の極限で発散し，初期値が定まらないために FMU の作成ができないことがわかる．なお，両端それぞれを 2 つの電圧出力アダプタで接続するような回路では，状態が定まらないために FMU の作成を行うことができない．

6.8.2 コンデンサを含む回路

図 6-43 にコンデンサを含む回路のモデルを示す．この回路の 2 端子対パラメータは，(6-52)式のように h-パラメータで表される．

$$\begin{bmatrix} e_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} z_{CG} & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ e_2 \end{bmatrix} \tag{6-52}$$

ここで，インピーダンス Z_{CG}は，

$$z_{CG} = \frac{R_1 R_2 + \dfrac{R_1 + R_2}{sC + G}}{R_1 + \dfrac{1}{sC + G}} \tag{6-53}$$

であり，$s\to\infty$の極限でインピーダンス Z_{CG}は直列等価抵抗 R_2に収束する．

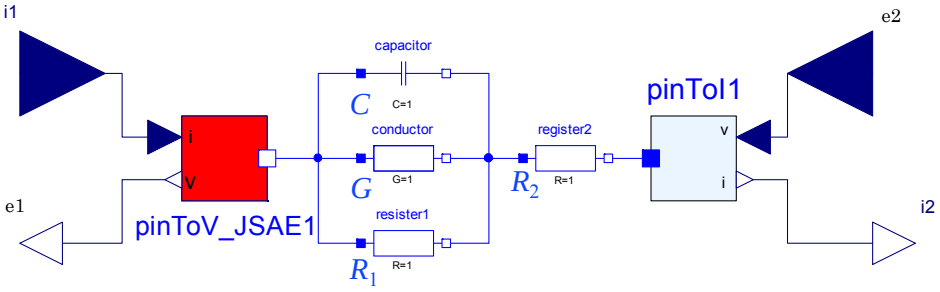


図 6-43　FMU 作成に用いたキャパシタ，コンダクタと抵抗のモデル

図 6-43 の回路の両端をいずれも電流出力アダプタにする場合には，(6-54)式のように Y-パラメータで表される．

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \frac{1}{z_{CG}} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{6-54}$$

(6-54)式では，(6-53)式に示すインピーダンス Z_{CG} を分母に持つため，直列等価抵抗 R_2 が 0 のとき s→∞の極限で発散する．このような回路では FMU が作成できないため，キャパシタを含む回路の出力を電流として FMU 作成する際には直列等価抵抗の有無に注意し，無い場合には R_2に相当する直列等価抵抗を追加するなどする必要がある．

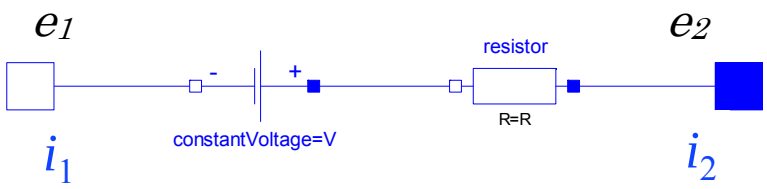
6.8.3　電圧源

図 6-44 (a)に電圧源と，それに直列に接続される抵抗 R で構成される回路の模式図を表す．図 6-44 (b)は，図 6-44 (a)のアダプタの接続をそれぞれ電流出力アダプタと電圧出力アダプタとし，ブロック線図を用いて因果的に構成する場合について示している．図中の e_{target} は，理想電圧源の電圧設定値を示す．図中の gain1 は理想電圧源をオペアンプで表した際のゲインであり，係数 k には大きな値を与える．図 6-44 (c)は，同じ回路のアダプタ接続をそれぞれ 2 つの電流出力アダプタとして，ブロック線図を用いて因果的に構成する場合について示している．図 6-44 (b)および図 6-44 (c)はそれぞれ，(6-55)式,(6-56)式のように表すことができる．

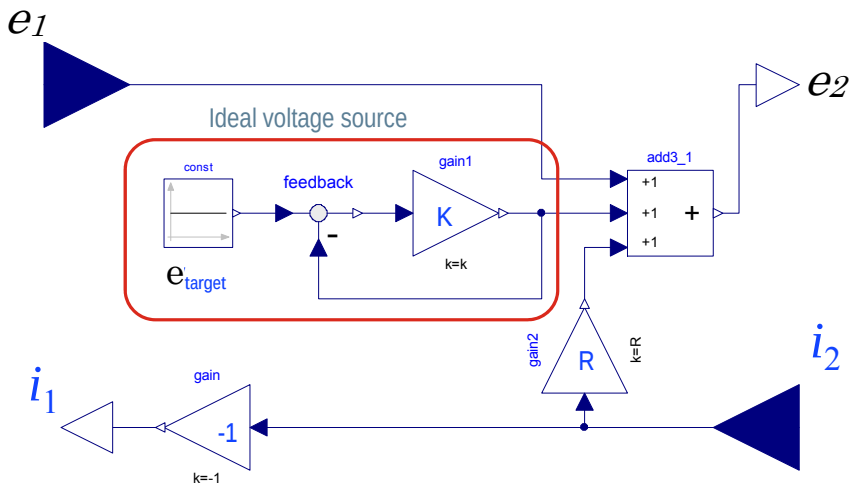
$$\begin{cases} e_2 = e_1 + Ri_2 + \dfrac{K}{1+K} e_{\text{target}} \\ i_1 = -i_2 \end{cases} \tag{6-55}$$

$$\begin{cases} i_1 = -\dfrac{e_2 - e_1 - \dfrac{K}{1+K} e_{\text{target}}}{R} \\ i_2 = \dfrac{e_2 - e_1 - \dfrac{K}{1+K} e_{\text{target}}}{R} \end{cases} \tag{6-56}$$

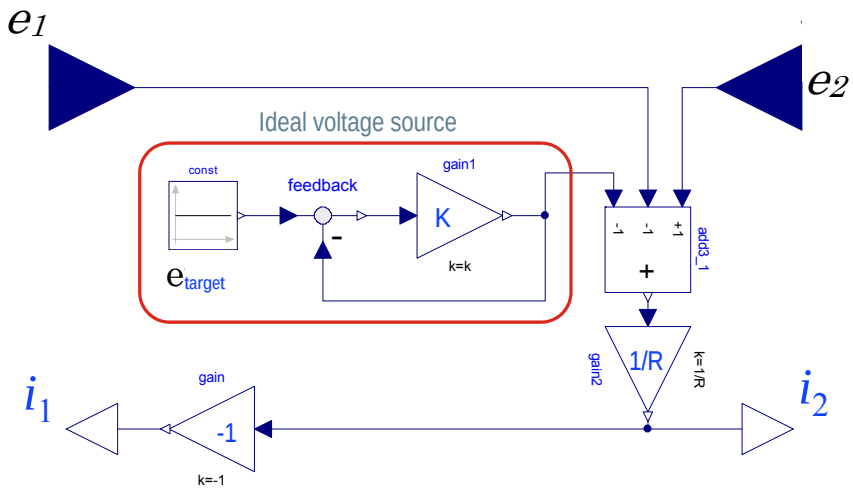
(6-55)式の右辺で R=0 のとき，回路は理想電圧源となり，e_2出力は電流値に依存しない．一方．(6-56)式の右辺では分母に抵抗 R を持ち，R=0 で電流値が発散するする．これより，理想電圧源の両端のアダプタをどちらも電流出力アダプタにして接続することはできないことがわかる．



(a) 理想電圧源と抵抗からなる回路

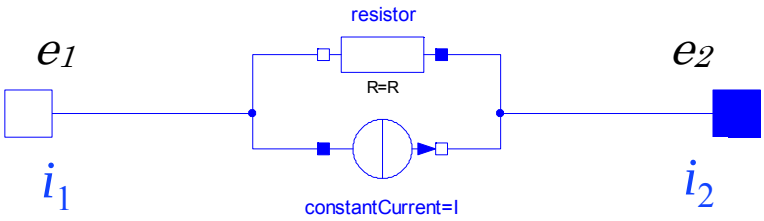


(b) h-パラメータを用いた因果的接続

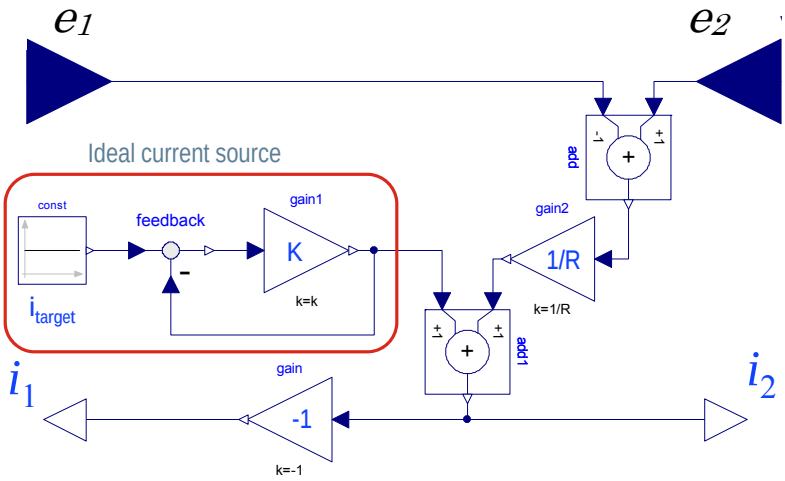


(c) Y-パラメータを用いた因果的接続

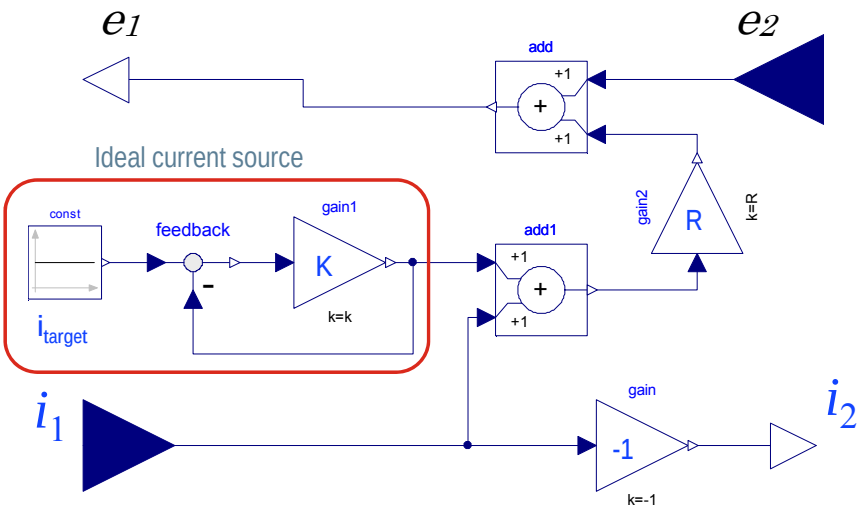
図 6-44　理想電圧源



(a) 理想電流源と抵抗からなる回路



(b) Y-パラメータを用いた因果的接続



(c) h-パラメータを用いた因果的接続

図 6-45　理想電流源

6.8.4 電流源

図 6-45 (a)に電流源と，それに並列に接続される抵抗 R で構成される回路の模式図を表す．図 6-45 (b)は，図 6-45 (a)のアダプタの接続をそれぞれ 2 つの電流出力アダプタにして，ブロック線図を用いて因果的に構成する場合について示している．図中の i_{target} は，理想電流源の電流設定値を示す．図中の gain1 は理想電流源をオペアンプで表した際のゲインであり，係数 k には大きな値を与える．図 6-45 (c)は，同じ回路のアダプタ接続をそれぞれ電流出力アダプタと電圧出力アダプタとし，ブロック線図を用いて因果的に構成する場合について示している．図 6-45 (b)および図 6-45 (c)はそれぞれ，(6-57)式, (6-58)式のように表すことができる．

$$\begin{cases} i_1 = -\left(\dfrac{e_2 - e_1}{R} + \dfrac{K}{1+K} i_{\text{target}} \right) \\ i_2 = \dfrac{e_2 - e_1}{R} + \dfrac{K}{1+K} i_{\text{target}} \end{cases} \tag{6-57}$$

$$\begin{cases} e_1 = e_2 + R\left(\dfrac{K}{1+K} i_{\text{target}} + i_1 \right) \\ i_2 = -i_1 \end{cases} \tag{6-58}$$

(6-57)式の右辺で抵抗 R を無限大とするとき，図 6-45 (b)の回路は理想電流源となり両端の電圧によらず設定した電流 itarget を出力する．他方，(6-58)式の右辺で抵抗 R を無限大とするときには，電流値は理想電流源の電流設定値 i_{target} を反映せず，両端の電位差は発散する．これより，理想電流源については，両端のアダプタの接続を，どちらも電流出力アダプタにする必要のあることがわかる．一方，内部抵抗を導入した場合は，一端を電圧出力，他端を電流出力として，FMU を作成することが可能であることがわかる．

6.9 FMU 接続時の注意点

非因果的接続に対応するモデリング言語を用いたツールでは，因果的接続に対応した FMU を，6.4 節で説明したアダプタを用いることで，非因果的接続を行うことが可能である．しかしながら，接続方法によってはシミュレーションの実行ができない場合や連成シミュレーションする場合には同期設定が必要となるため，本節ではその注意点について述べる．

6.9.1 シミュレーション結果が発散する場合

FMU では，入出力の方向がどちらか一方向のみに決まっている．このため，非因果的接続の可能なツールでは，アダプタを用いることで，入出力の方向を気にせず使用できるこ

とが期待される．しかしながら，複数の FMU を組み合わせて使用する際，アタプタを用いても，組み合わせによってはシミュレーションの実行が行えない場合が存在する．図 6-42 と図 6-43 に示した回路で作成した 2 つの FMU の電流と電圧の入出力の向きが互いに向き合うように接続している場合について，回路図を図 6-46 に示す．ここでは，電流出力アダプタのみを用いて 2 つの FMU 間を接続している．

このような電流・電圧の方向が一致していない箇所のある回路でシミュレーションを行う場合，Co-Simulation モードでは，シミュレーションを実行することができない場合がある．図 6-47 (a)のように，2 つの FMU の間に 0.1Ω の低抵抗を挿入した場合，Co-Simulation モードでもシミュレーションの実行が可能になるが，図 6-47 (b)に示すように電圧値・電流値が $\pm 10^{39}$ 程度のオーダーまで大きく振れ，発散していることがわかる．

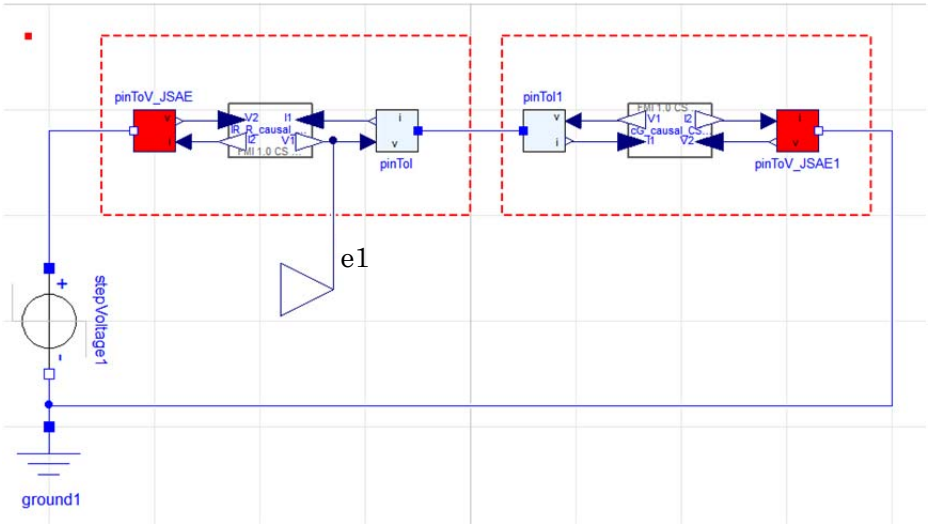
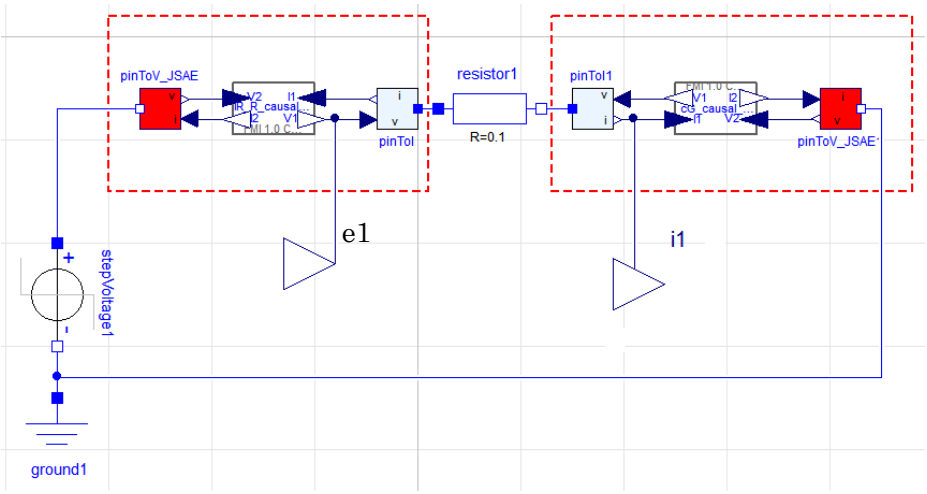
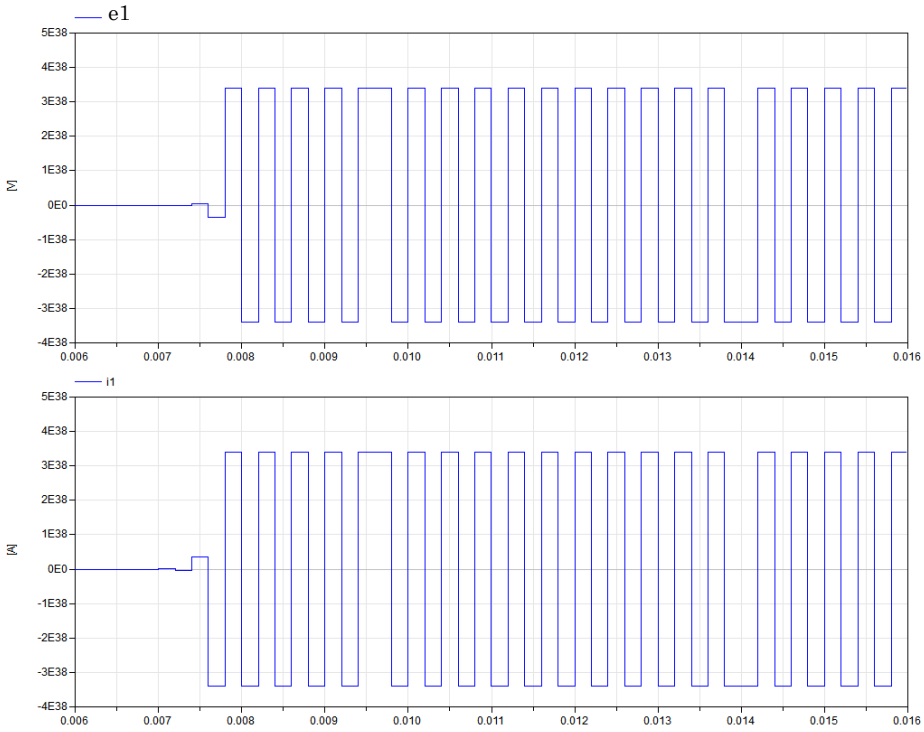


図 6-46　“structually singular error”の発生するシミュレーション回路



(a) シミュレーション回路



(b) シミュレーション結果 – 上段: e_1, 下段: i_1

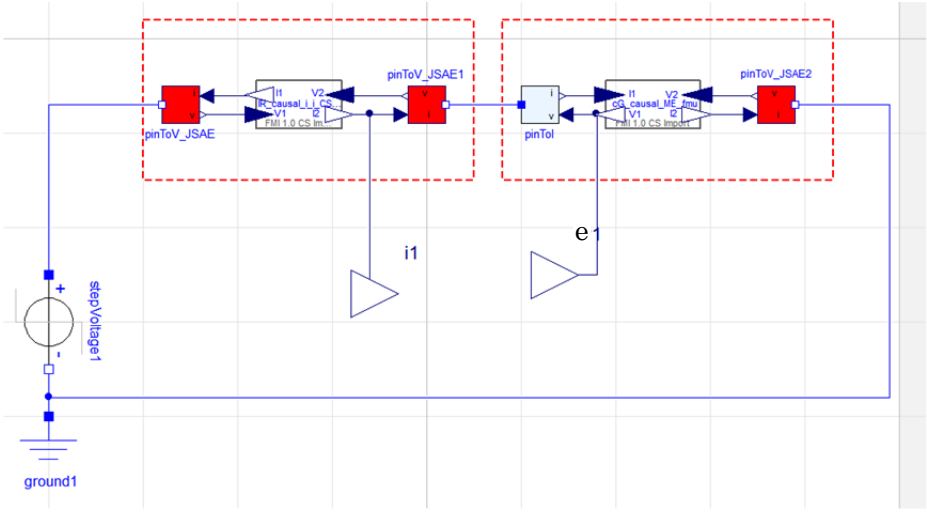
図 6-47　シミュレーション結果が発散する場合

6.9.2　シミュレーション結果が収束する場合

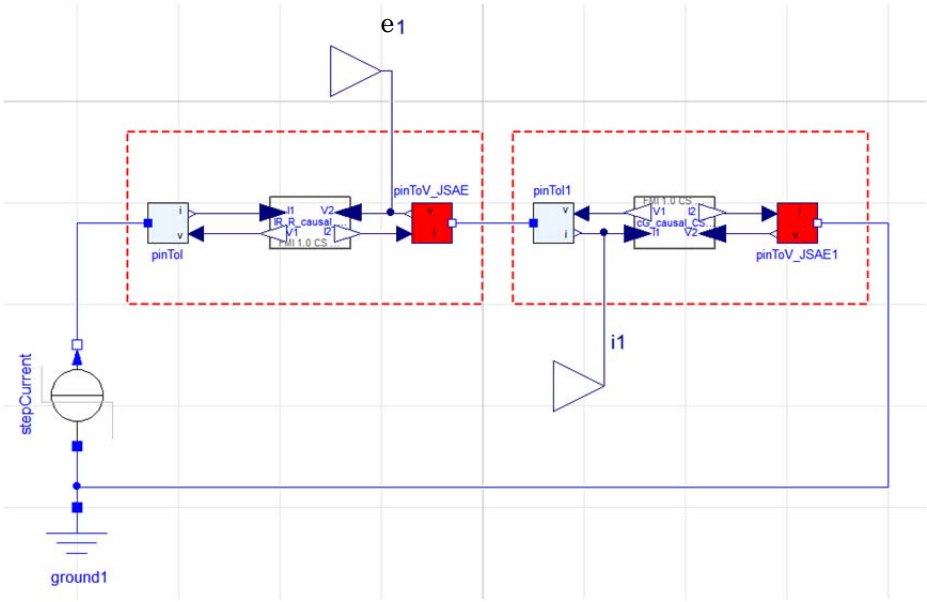
次に, FMU が Model Exchange モードおよび Co-Simulation モードの両方でシミュレー

ションの実行が可能な接続方法について示す．6.9.1 節で示したような発散が生じる場合には，この節で示すような接続にすることで，シミュレーションの実行が可能になる．

図 6-48(a),(b)の回路図は，図 6-42 と図 6-43 に示した回路で作成した 2 つの FMU を非因果的に接続したものである．これらの回路では，電圧・電流の方向が，どちらも一方向になるように接続している．



(a) 電圧入力のとき



(b) 電流入力のとき

図 6-48　接続アダプタを用いて FMU を非因果的に接続したシミュレーション回路

図 6-48 (a)では図 6-42 (c)の FMU を用いて電圧を入力しており，電圧入力に対するステップ応答特性をみることができる．図 6-48 (a)の電圧源の出力電圧と出力電流をそれぞれ e_1, i_1 とするとき，e_1 と i_1 の関係は次式のように表される．

$$e_1 = \left(z_{LR\_R} + z_{CG}\right) i_1 \tag{6-59}$$

ここで，Z_{LR_R}および Z_{CG}は，(6-51)式および(6-53)式に示したインピーダンスである．(6-59)式の次元は，FMU で構成される系の入力の次元に合わせている．(6-59)式の右辺は $s\to\infty$のとき収束し，初期値を求められることがわかる．

図 6-48 (b)では図 6-42 (b)の FMU を用いて電流を入力し，電流入力に対するステップ応答をみることができる．図 6-48 (b)の電流源出力電流と出力端電圧をそれぞれ e'_1, i'_1 とするとき，これらの関係を電流の次元で示すと，(6-60)式のように表される．

$$\left(1 + \frac{z_{CG}}{z_{LR}}\right) i'_1 = \frac{1}{z_{LR}} e'_1 \tag{6-60}$$

Z_{LR}および Z_{CG}は，(6-52)式および(6-53)式に示したインピーダンスである．(6-60)式の右辺もまた，$s\to\infty$のとき収束し，初期値を求めることが可能である．

6.9.3 FMU が電圧源・電流源を含む場合

バッテリなどの複数電圧源のマネジメントについてシミュレーションを行う場合には，複数の電圧源を互いに接続する必要がある．しかしながら，電圧源を含む FMU を，接続アダプタを介して複数個並列に非因果的接続する場合，Model Exchange モードではシミュレーションを実行できるものの，Co-Simulation モードでは Structurally Singular エラーが発生することが確認されている．

このようなケースでは，各々の電圧源の出力端に直列に低抵抗を付加する，あるいは電圧を出力する FMU を一つだけにするなどの工夫が必要になる．また，各 FMU 間の通信間隔の設定が同じであると，Co-Simulation モードの同期がずれて FMU の出力が干渉し，発振するケースが存在する．このようなときには，通信間隔を個々に設定することで，電源間の干渉が避けられる場合がある．

6.9.4 分割モデル間の同期について

一般的に異なるツール同士で連成シミュレーションする場合，ソルバや時間刻みに注意する必要がある．可変時間刻み法の場合は各ツールにおいて数値積分アルゴリズムごとに異なる可変時間刻みアルゴリズムが採用されるため，シミュレーション結果が異なる場合がある．

(1) VHDL-AMS または Modleica でモデルを統一した場合

各ツールで使用するソルバを同じものにして時間刻みを小さくすることで同期ずれを防

止できることは検証済みである [14].

(2) FMI の場合

FMI の場合，Model Exchange と Co-Simulation で状況は異なる．Model Exchange でモデルを１つのツールに統合した場合は，ソルバは自動的に同一になるため，同期についてはモデルをインポートしたツールの内部時間刻みに応じて連続的に変化する．一方 Co-Simulation ではモデル間の同期については通信間隔で離散的に同期を取るため，オリジナルのシミュレーション波形に近づけるためにはこの数値を小さくすることで同期ずれを小さくすることができる（図 6-49）.

また， Co-Simulation マスタの実装によっては，通信間隔による遅延は FMU ごとに累積されていくため，どう FMU に分割するかで結果が変わることがあるので注意が必要である．図 6-50 は振幅１V の信号源にバッファを 4 段接続した回路だが，オリジナルのツールによる結果に対し，通信間隔を 1ms に設定して FMU 化すると，4 段のバッファにより 1ms ×4＝4ms の遅延が生じていることがわかる.

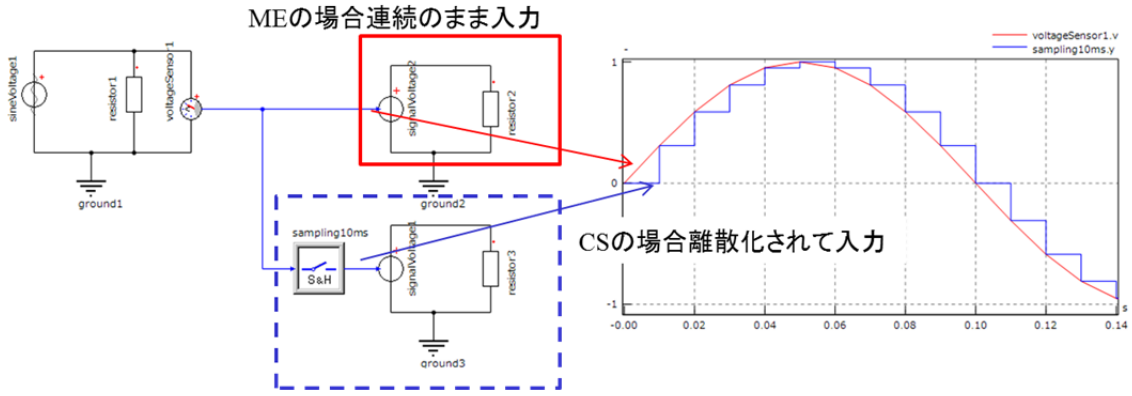
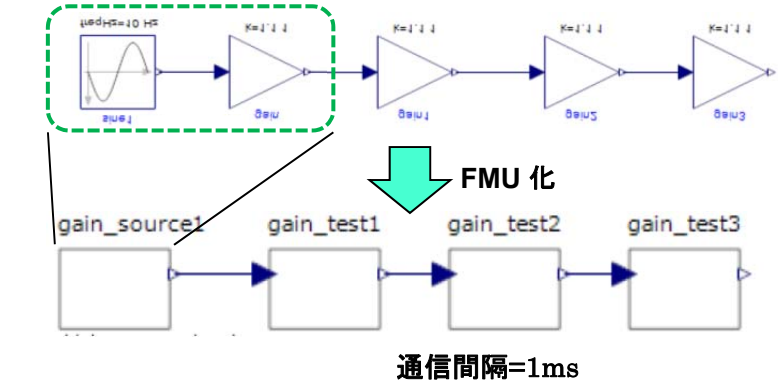


図 6-49 CS と ME の同期の違い



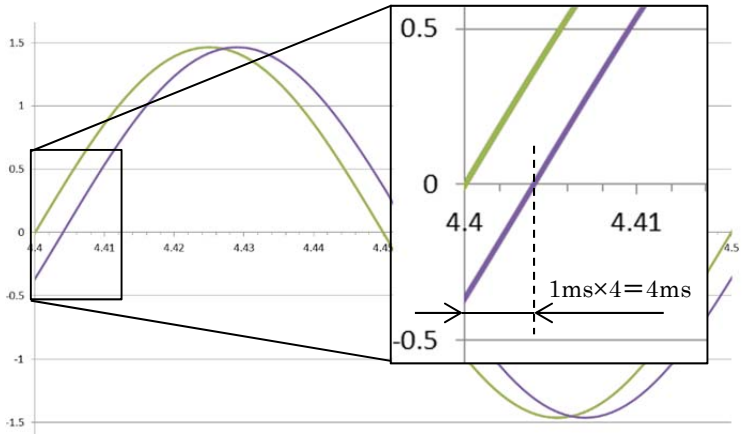


図 6-50 FMU 直列接続による遅延時間の累積

第7章　用語説明

・　Co-Simulation

FMI で規定されている二つの接続方法の一つで, FMU 側にソルバを持つ場合を指す. 5.2.1 参照．一般的には異なるツール間で通信をしながら同時にシミュレーションを行うことを言う.

・　FMU

Functional Mockup Unit の略称. FMI 仕様に基づくモデル交換（Model Exchange）と連成（Co-Simulation）のためのモデル計算モジュールや入出力など関連情報を zip 形式にまとめたもの.

・　Model Exchange

FMI で規定されている二つの接続方法の一つで, FMU 内部にソルバを持たない場合を指す．5.2.1 参照.

・　Modelica

微分代数方程式を利用したシステムモデリングのためのオブジェクト指向の言語. 言語仕様は Modelica 協会によってとりまとめられ，1997 年に Ver. 1.0，2014 年 10 月現在での最新版として Ver. 3.3 Release1 がリリースされている. モデリングのため, オープンソースの Modelica Standard Library (MSL)が提供されている.

・　Modelica 協会

モデリング，シミュレーション，物理的および技術的なシステムやプロセスのプログラミングなどを用途として，Modelica の開発，普及促進を目的とする非営利，非政府組織．トレードマーク，Modelica の言語仕様，MSL など無形の権利を有し，管理している．これらは，産業の発展と研究の推進のために広く一般に利用公開されている.

・　MultiRate Modeling

スティフ系の計算において，時定数の大小に応じてモデルを分割し，数値積分の時間刻みを，それぞれのサブモデルの時定数に応じて設定して計算する方法.

・　VHDL-AMS

VHDL はデジタルシステムのためのハードウェア記述言語であり，VHDL-AMS (Analog and Mixed-Signal)はデジタルおよびアナログシステムのハードウェア記述に対応した VHDL の拡張スーパーセット．（IEEE std.1076.1-1999, IEC 61691-6）

・　アクロス変数

物理モデリングにおいて，スルー変数と対で用いられる変数．電気回路における電圧のように, 2 点間の差を表し, 回路中の任意の一閉路に沿ってその差を積算したとき, 一巡すると必ず 0 になる．機械系における速度や角速度（あるいは変位や角度），熱流体系における圧力や温度などがこれに相当する.

・　因果的接続

入力量と出力量をあらかじめ指定したモデル作成手法における要素間の接続．入出力量の伝達方向は 1 方向となる.

・ スルー変数

物理モデリングにおいて，アクロス変数と対で用いられる変数．電気回路における電流のように，ある点を通過する量を表し，回路中の任意の一点に流れ込む（あるいは流れ出す）量の総和は必ず 0 になる．機械系における力やトルク，熱流体系における質量流量や熱流量などがこれに相当する．

・ スレーブ

Co-Simulation において FMU を作成するツール．作成された FMU はソルバが組み込まれて単独で実行できる形式と，ソルバが組み込まれず実行するためにはスレーブモデルを作成したソルバを持つツールが必要な形式がある．

・ ソルバ

微分代数方程式(DAE)や常微分方程式(ODE)を数値解法で解くプログラム．各種シミュレーションツールでは，何らかのソルバが実装されている．数値計算エンジンとも言われる．扱う系の性質（DAE か ODE か，系に含まれる動特性の時定数が大きく異なるスティフ系かどうか，イベント処理があるかどうか，など）により，適切なソルバのアルゴリズムと，時間刻みや収束計算の許容精度などを選ぶ必要がある．

・ 通信間隔（communication step sizes）

Co-Simulation の FMU がマスタと変数の入出力を行う時間の間隔のこと．FMU はこの時間間隔の間にそれぞれのモデルの計算を行う．

・ 非因果的接続

回路図のような物理的事象のつながりをモデリングする場合の物理要素間の接続．接続線の持つ情報にはスルー変数とアクロス変数があり伝達方向は接続状態により変化する．異なる物理領域ごとの物理要素接続となる．

・ マスタ

Co-Simulation において FMU を読み込み実行するツール．スレーブモデルの時間管理も行うがソルバは必ずしも必要ではない．

第8章　引用文献

1. **辻公壽ほか.** 自動車システム開発のためのモデルの要件と適用，Vol. 4, 4-S13, p.15-18.　：　, 24 年電気学会全国大会シンポジューム講演 4-S13-5.
2. 微分代数方程式に基づく非因果的物理モデリング. **平野　豊.**　：計測自動制御学会, 2014 年, 第 53 巻.
3. Dymola Users Manual Volume1.
4. **Synopsys.** HSPICE® Simulation and Analysis ユーザガイド　Ver2004/0. 2004.
5. **MathWorks.** (オンライン) http://www.mathworks.co.jp/support/solutions/ja/data/1-9J89SL/index.html?solution=1-9J89SL.
6. **MathWorks.** (オンライン) http://www.mathworks.co.jp/jp/help/simulink/ug/simulating-dynamic-systems_ja_JP.html.
7. **FAT-AK30 (Working Group: Simulation of Mixed Systems with VHDL-AMS).** (オンライン) http://fat-ak30.eas.iis.fraunhofer.de/.
8. **Peter J. Ashenden ほか.** The System Designer's Guide to VHDL-AMS: Analog, Mixed-Signal, and Mixed-Technology Modeling.　: Morgan Kaufmann, 2002.
9. **Ulrich Heinke l ほか.** The VHDL Reference: A Practical guide to Computer-Aided Integrated Circuit Design including VHDL-AMS.　: Wiley, 2000.
10. **Elmqvist H. ほか.** Fundamentals of Synchronous Control in Modelica. Munich, Germany : Proceedings of 9th International Modelica Conference, 2012-9.
11. **Torsten Blochwitz ほか.** The Functional Mockup Interface for Tool independent Exchange of Simulation Models. (オンライン) 2011 年年. https://trac.fmi-standard.org/export/700/branches/public/docs/Modelica2011/The_Functional_Mockup_Interface.pdf.
12. **Modelica Association.** Functional Mock-up Interface for Model Exchange and Co-Simulation Document version:2.0. (オンライン) 2014 年年. https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf.
13. **パワーエレクトロニクスシステムにおけるモデリングとシミュレーション技術共同研究委員会編.** パワーエレクトロニクスシステムにおけるモデリングとシミュレーション技術，電気学会技術報告 第 1114 号.　：電気学会, 2008 年.
14. **嶋田敏ほか.** シミュレーションモデル接続技術の現状と課題　文献番号：20135494.　：JSAE, 2013 年 5 月.

第9章　索引

以上