

# MBD とは

## モデルベース開発 (MBD)

モデルベース開発は組み込み制御システムの開発にモデルを利用する開発手法です。

組み込み制御システムはハードウェアとソフトウェアを組み合わせてシステムが表現されます。そのため、開発者にはハードウェアとソフトウェア両方の知識と技術が求められます。実際の開発において、両方の知識と技術を持つ技術者を必要な人数分確保することは難しく、複数の部署や担当者に分担して開発されます。組み込み制御システムの分担は一般的に、

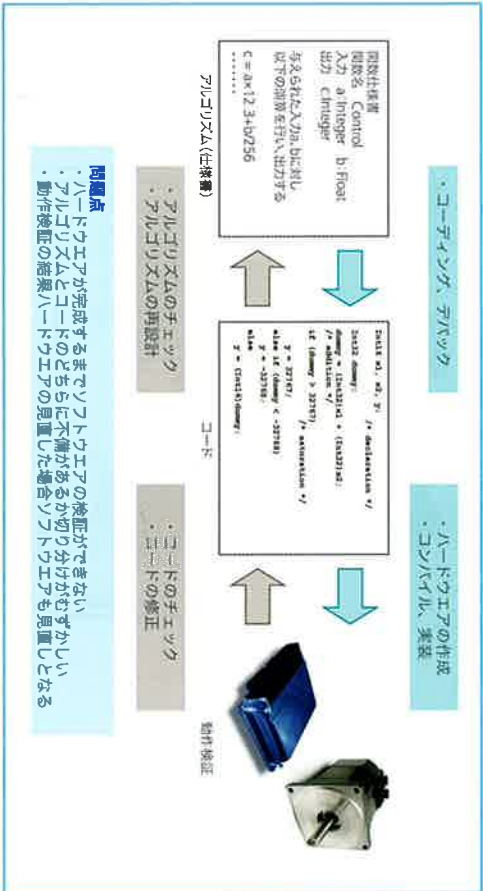
- 全体の仕様決め、設計を行うシステム開発ソフトウェアの開発
- ソフトウェアの設計・開発をおこなうソフトウェア開発

- ハードウェアの設計、開発をおこなうハードウェア開発

に分かれます。それぞれ分担して開発を進めていく上で、設計したソフトウェアアルゴリズムや試作ハードウェアを動作検証する必要があります。組み込み制御システムの特性上、ソフトウェアとハードウェアが両方そろわないと動作させることが

できませんが、お互いの作業が終わるまで待っていては効率よく開発を進めることができません。特にソフトウェアはCPUやI/O回路を搭載したコントローラ、センサやモーターアクチュエータ、そして制御対象のメカニズムがそろわないと動作させることができず、開発の早い段階で動作検証を行うことは難しいと言えます。また、ソフトウェアを動作させるためには設計したアルゴリズムをもとにコーディングを行う必要があります。アルゴリズムを変更することにコーディングをやり直す必要があるため、動作検証を行うための準備に時間がかかってしまいます。

もし動作検証によってアルゴリズムに問題があった場合、再度設計、コーディング、場合によってはハードウェアの設計直しなどの手戻りが発生します。問題の発見が早ければ早いほど開発の手戻りにかかる工数や費用は少なくなるため、可能な限り早く問題を発見できるように検証を行うことが望まれます。



従来の問題点

## モデルベース開発のメリット

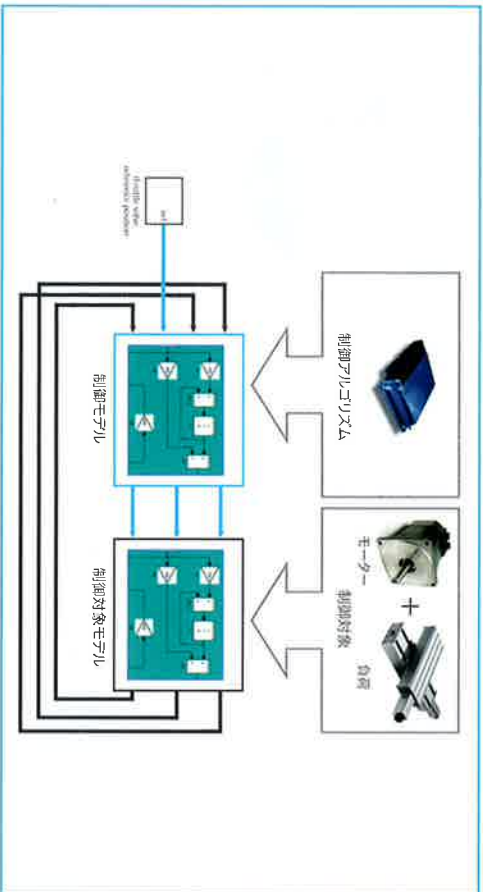
そこで、開発の早い段階で設計した制御アルゴリズムを検証するために、ハードウェア無しで検証する手法としてコンピュータシミュレーションが考えられます。コンピュータシミュレーションを行うためには、制御対象の挙動やソフトウェアアルゴリズムをコンピュータで計算できるように数式化されたモデルで記述する必要があります。このモデルを利用して、設計・検証を行ういくつかの手法がモデルベース開発です。

モデルベース開発では、シミュレーションによってソフトウェアとハードウェアを組み合わせた挙動を手軽に検証することが可能です。シミュレーションによる検証はハードウェアが不要というメリット以外にもさまざまなメリットがあります。

- 検証に必要な状況や環境を再現することが可能
- 同一条件での検証を再現することが容易
- モデルを共有することで、複数の開発者に検証環境を用意できる

開発対象が複雑化し、検証項目が増えていっている昨今において、上記のメリットはますます重要になってきています。

ソフトウェアアルゴリズムをモデルで記述することで、コーディングをする前にアルゴリズムの検証を行うことができます。コーディング後に問題が見つかった場合、原因がアルゴリズムの設計不具合なのか、またはコーディングミスなのかを見極めるのに時間がかかりますが、先にアルゴリズム検証を済ましておき、モデル自体をソフトウェアの仕様書とすることで、作業分担や問題の切り分けが容易になります。

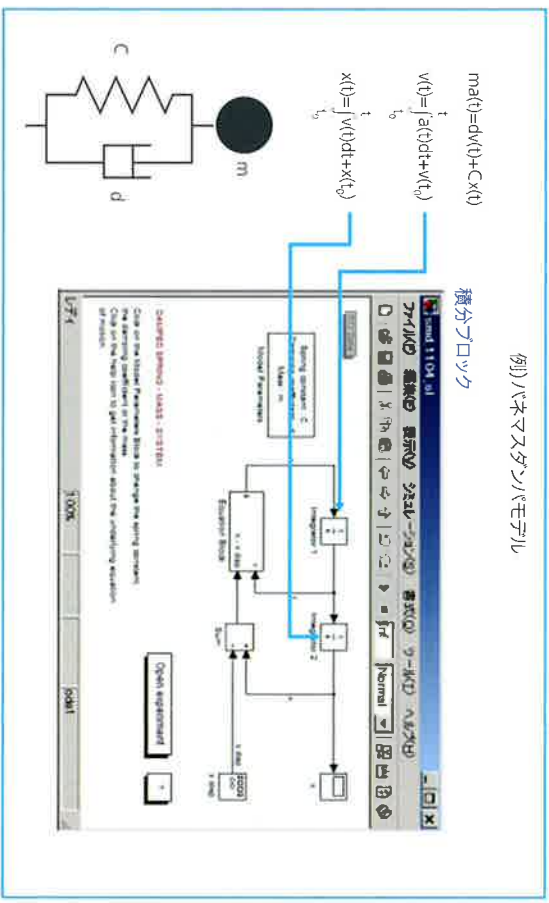


モデルを用いたシミュレーションで検証

## モデルベース開発ツール

モデルベース開発を効率よくおこなうためにはシミュレーション環境も重要になります。運動方程式などの物理式と制御で利用するアルゴリズムを組み合わせてシミュレーションを行う必要があり、一般的には設計・シミュレーションツールを利用します。モデルベース開発における代表的なツールとして The Mathworks 社の MATLAB/Simulink が上げられます。高度な数値計算をコマンド形式で行う MATLAB と、ブロック線図でグラフィカルに数式やアルゴリズムを記述してシミュレーション可能な Simulink の組み合わせで、モデル作成とシミュレーションが容易に行える環境となります。Simulink

は積分や微分などもブロックが用意されており、運動方程式の計算も容易に行えます。また、Simulink で制御アルゴリズムのモデル作成をすることで、コーディングの知識・経験がなくてもアルゴリズムの設計と検証が可能になります。また、アルゴリズム開発者を集めることが容易になります。また、フックリポートモードで良く使われる色々な運転モードを遷移するような制御ロジックを見やすくモデル化できる Stateflow を使うことも可能です。Stateflow は Simulink の拡張で、ステートマシンとフローチャートを使い制御ロジック、モードロジックを容易にモデル化できます。



Simulink シミュレーションの例

# ラビットコントロールプロトタイプング (RCP)

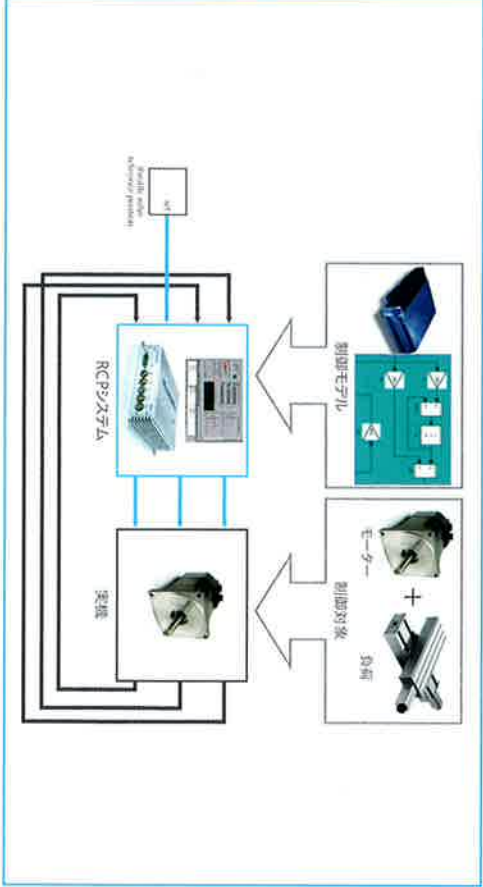
## 実機による検証の重要性

制御対象をモデルで記述し、シミュレーションによって開発の初期段階から検証を行うことはできますが、すべての検証をシミュレーションのみで行うことはできません。制御対象を数式で記述することは実物と同一の挙動を得ることができれば理想的ですが、現実には数式化できない挙動やパラメータ誤差などで実物とは挙動が異なります。そのため実物を使った検証は必ず必要になります。実機を使った精度の高い検証を可能な限り早い開発段階で行うために、制御アルゴリズムを検証するための試作コントローラを用意するのが一般的です。

モデルベース開発の場合、制御アルゴリズムのモデルを元に試作コントローラを素早く用意する、ラビットコントロールプロトタイプング (Rapid Control Prototyping, 以後 RCP と略す) という手法があります。

## RCP ツール

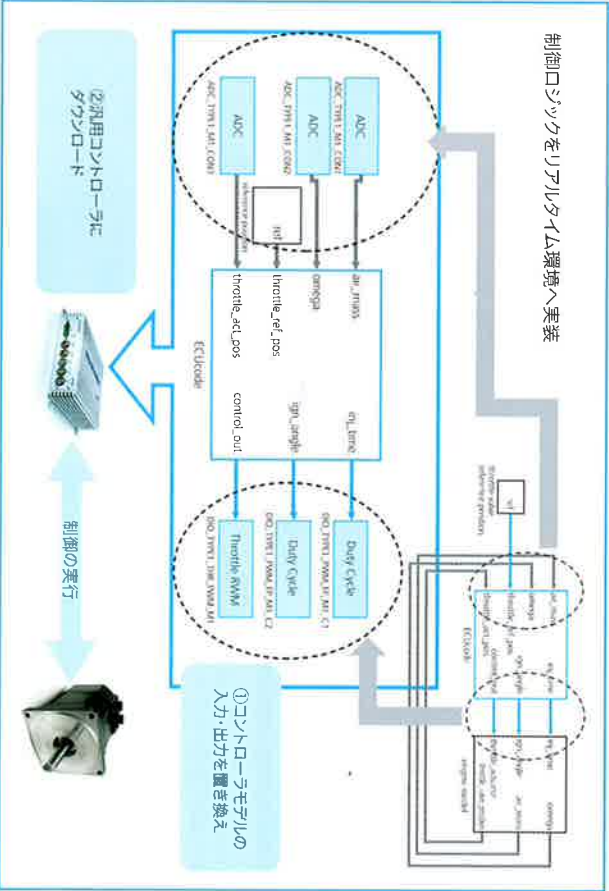
RCP は、製品版のコントローラでは利用しない高性能な CPU とコントローラによく搭載される I/O を一通り備えた汎用的な試作コントローラを用意し、制御アルゴリズムのモデルを素早く実装可能なソフトウェア環境と組み合わせることによって実現されます。これにより RCP は、専用の試作コントローラを作成することやコーディングを行うこと無しに、制御アルゴリズムの検証を行うことができます。



RCP システムと実機で検証

dSPACE の RCP システムは Simulink と高い親和性を持っており、コントローラ I/O にアクセスするための Simulink ブロックライブラリや Simulink と連携してボタンひとつで試作コントローラに実装する環境を用意しています。

I/O の種類も一般的な AD、DA、デジタル入出力、PWM 入出力、RS232 などのシリアル通信や、エンコーダ、シリアルバ、3/6 相 PWM などモーター制御に必要な I/O もそろえています。

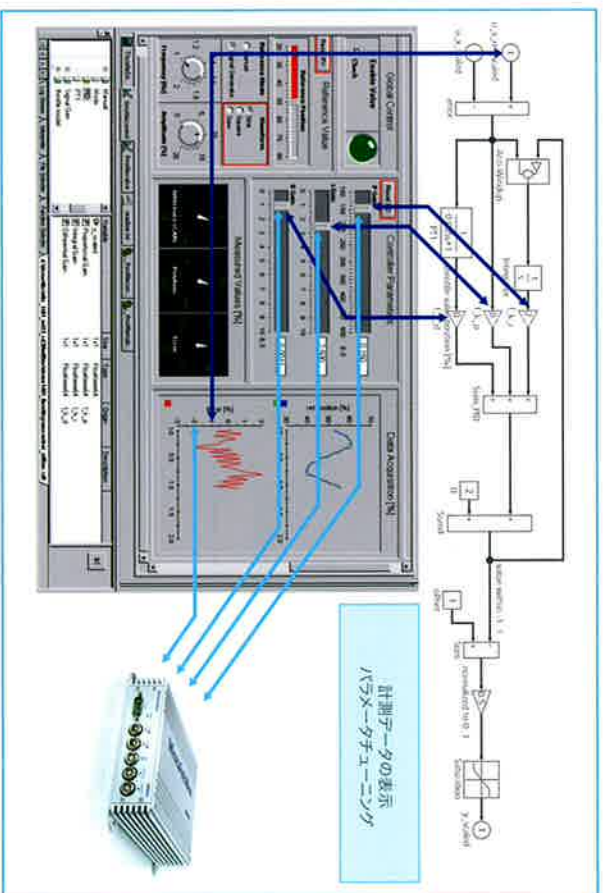


モデルをすぐに実装可能

また、実装した制御アルゴリズムが期待通りに動作しているかを検証するためには、コントローラの計測値や計算結果を把握し、制御パラメータをチューニングする必要があります。dSPACE は制御実行中の試作コントローラから計測データを取得し、PC 画面にグラフィカル表示するソフトウェアとして ControlDesk を用意しています。またこの ControlDesk から制御パラメータの変更も実行中にリアルタイムで行

えるため、短時間で設計検証を行うことが可能です。

このように RCP システムは実物を使った検証を素早く気軽に行えるため、制御対象の数式化やシミュレーションモデルの作成が難しい複雑なシステムに対する開発にも有効です。実物を使ってトライ＆エラーを繰り返し、効果的な制御アルゴリズムや制御パラメータを設計していくことができます。またコーディング無しに



動かしながら計測・パラメータチューニング

コントローラを用意できるため、センサ、モーター、機構設計などのハードウェア開発における検証ツールとして、ハードウェア開発者の方にもご利用いただけます。



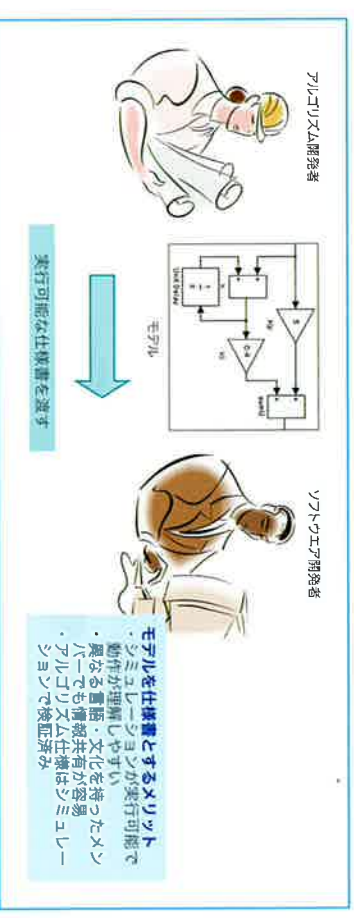
## 自動コード生成

### モデルとコードの違い

シミュレーションやRCP手法で検証されているのは純粋な制御アルゴリズムですので、ソフトウェアの設計や検証を行うフェイズが必要です。モデルによって設計・検証された制御アルゴリズムを製品となるコントローラに実装する際には、コントローラの特性に合わせたコーディングを行う必要があります。例えばSimulinkモデルで作成された制御アルゴリズムはシミュレーション精度を重視して浮動小数点演算で設計されますが、コントローラは価格を抑えるために固定小数点演算のみ可能なプロセッサを利用するケースが一般的です。またメモリ容量やプロセッサ速度も制限され、その中

で十分な性能を発揮するソフトウェアを設計することが求められます。

モデルベース開発ではアルゴリズム設計とソフトウェア設計を明確に分けることができるため、ソフトウェア開発を専門の部署に依頼するケースや、他社にアウトソーシングすることが比較的容易になります。その際に作成したモデルを実行可能な仕様書として扱うことができます。モデルは言葉では曖昧になりがちなソフトウェア仕様をシミュレーション結果として明確に伝えることができます。また言語に依存せず仕様を伝えられるため、海外の開発者とコミュニケーションをとることも容易になります。

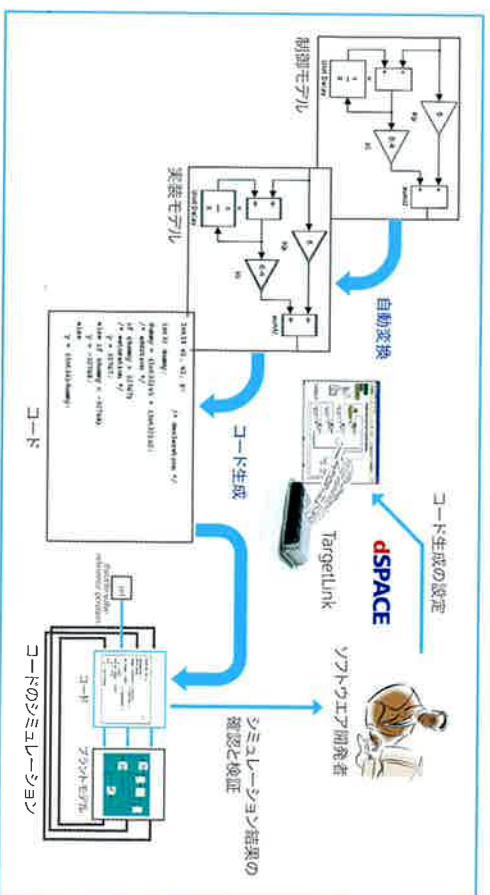


モデルを実行可能な仕様書に

### 自動コード生成ツール

実行可能な仕様書を元にソフトウェア開発者がモデルを見ながらコーディングを行う限りタイプライターや仕様の読み違いなどのヒューマンエラーは少なからず起きてしまいます。これに対して事前に検証した制御アルゴリズムのモデルを利用し、ソフトウェア開発者が効率よくソフトウェア設計や検証を行えるようにする手法として自動コード生成 (Auto Code Generation、以後ACGと略す) が生まれました。ACG手法

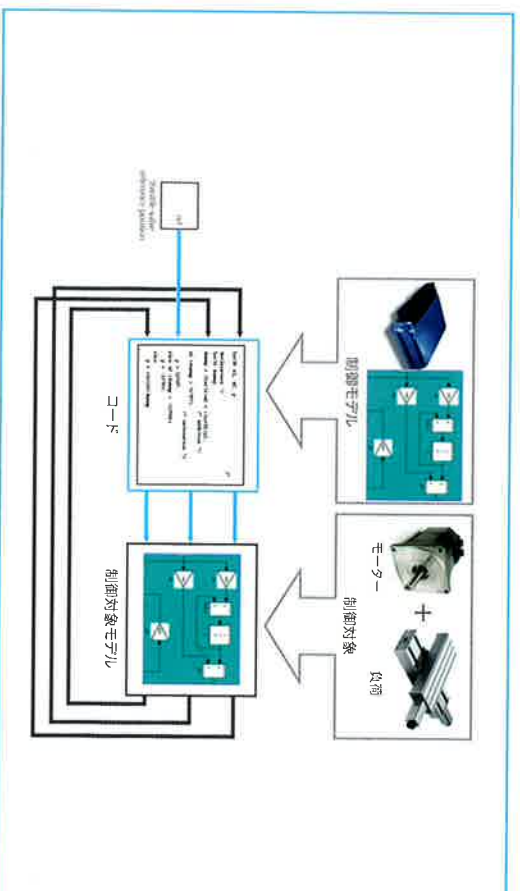
は専用のツールを利用してモデルからコードを自動で生成します。ACGツールがモデルを解析してコードを生成するため、ソフトウェア開発者は制御アルゴリズムをコードで実現することと工数を割く必要はありません。ただしモデルはコントローラの特性が考慮されていないため、ソフトウェア開発者が最適なコード生成のためのコーディングルールや設定を行っていきます。



モデルからコードを生成する流れ

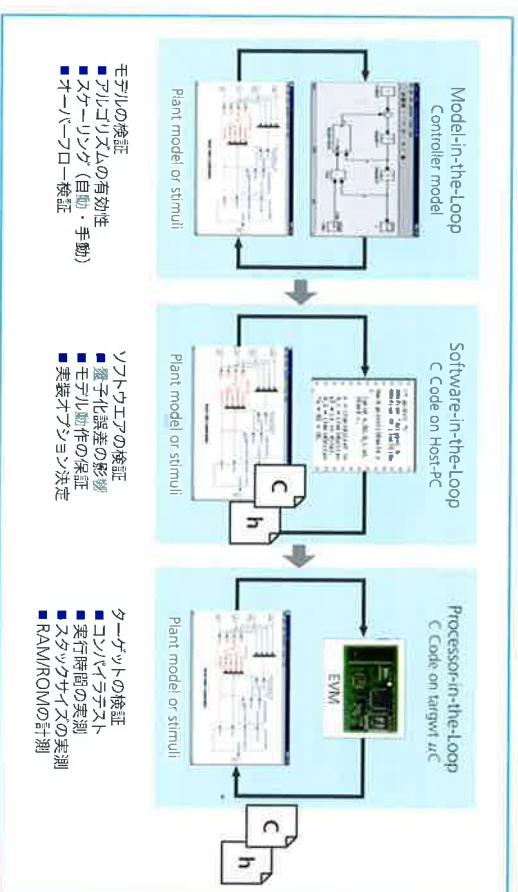
dspace では Targetlink という ACG ツール を用意しています。Targetlink は複雑な Simulink モデルであってもモデル構造を解析して、経験豊富なソフトウェアエンジニアに匹敵する高効率なコードを自動生成することが可能です。また

生成したコードと制御アルゴリズムの検証で利用したモデルを組み合わせ、シミュレーションによるソフトウェアの検証を行うことができます。



コードをシミュレーションで検証

制御アルゴリズムモデル+制御対象モデルのシミュレーション (Model in the Loop シミュレーション、以下 MIL と略す) と Targetlink 生成コード+制御対象モデルのシミュレーション (Software in the Loop シミュレーション、以下 SIL と略す) を自由に切り替えることができるので、コードの動作検証や固定小数点化の影響などをシミュレーション結果から比較検討することが出来ます。またマイコンの評価ボード+制御対象モデルのシミュレーション (Processor in the Loop シミュレーション、以下 PIL と略す) によってより実際のコンローラに近い環境でシミュレーション検証を行うことも可能です。



モデルの検証  
■ アルゴリズムの有効性  
■ スケーリング (自動・手動)  
■ オフ・ベンチ検証

ソフトウェアの検証  
■ 電子化誤差の影響  
■ モデル動作の保証  
■ 実機オプション決定

ターゲットの検証  
■ コンパイラテスト  
■ 実行時間の実測  
■ スタックサイズの実測  
■ RAM/ROM の計測

複数のシミュレーション検証で高品質なコードに

Targetlink は MIL, SIL, PIL の素早い切り替えが可能のため、シミュレーションによる検証と設定変更を繰り返すことで高品質なコードを効率よく生成することが出来ます。



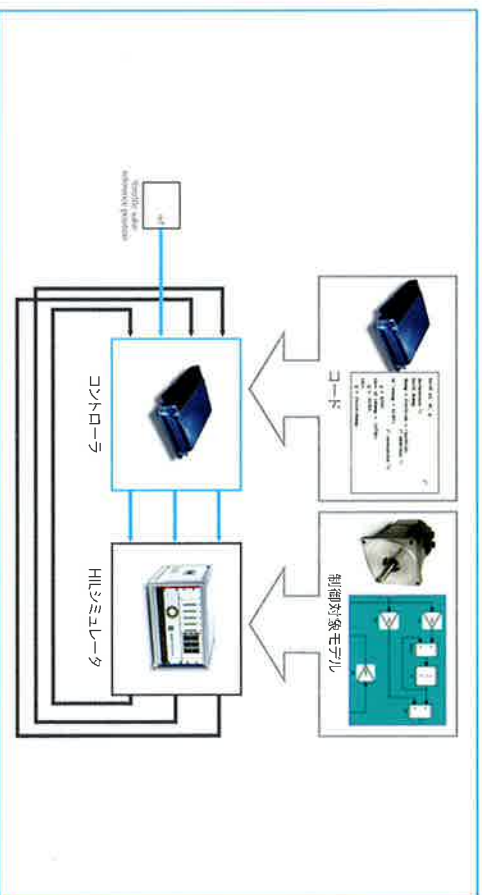
## Hardware In the Loop シミュレーション

## ハード依存ソフトウェアの検証

一般的には、コントローラに実装する全てのソフトウェアをすべてモデルで設計してACGツールでコード生成することはありません。I/Oや通信を行うためのドライバソフトウェアは主にハンドコードや提供されているライブラリを利用する場合が大半です。ドライバソフトウェアはI/Oや通信のハードウェアを動作させなければ動作検証ができないため、どうしても製品版のコントローラが出来上がった後に検証を行う必要があります。

## HILシミュレーション

この検証を効率よく行うためにHILシミュレー  
ション (Hardware In the loop シミュレーショ

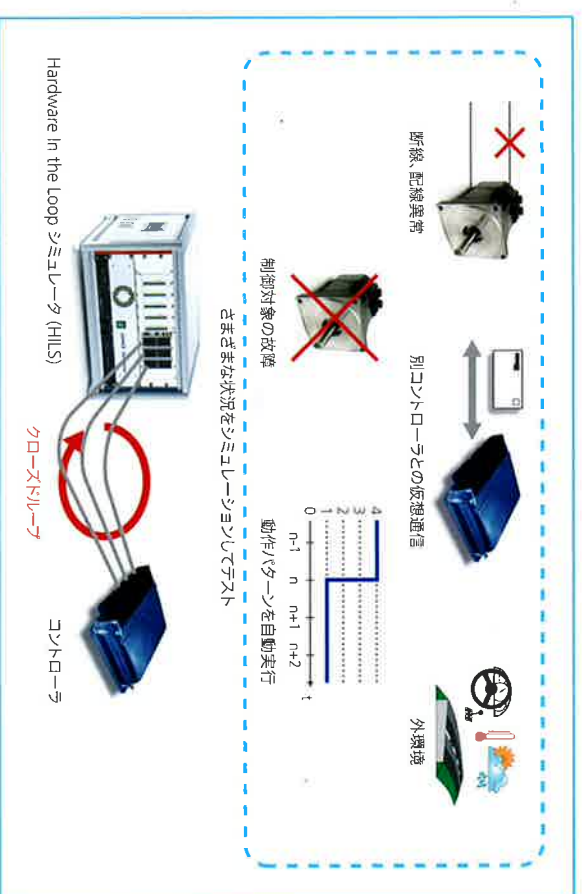


シミュレータによる仮想環境でコントローラ検証

ン)が有効です。コントローラに接続されたリアルタイムシミュレータは制御対象のメカニズムをシミュレーションし、センサ電圧の疑似出力やコントローラからの制御信号解析をシミュレーションに同期して行うことで、あたかもコントローラに本物の制御対象が接続されているかのように振舞います。これにより、以前に紹介したシミュレーションによる検証のメリットを、ハードウェアを含んだ検証でも享受できます。またリアルタイムシミュレータ内で計算されるシミュレーションモデルをアルゴリズムやソフトウェアの設計フェイズと共有することで、検証結果の比較が可能になり、問題発見時に原因がどこにあるかを検討しやすくなります。

dspaceのHILシミュレータはコントローラを接続するための豊富なI/Oを用意しています。またコントローラが接続された状態で断線・天絡・地絡・線間ショートを再現する欠陥生成シ

ミュージシャンや電源電圧の変動をシミュレーションするシステムなどを組み合わせたさまざまなテストを行うための環境をご用意いたします。



HILシミュレータを利用したテスト

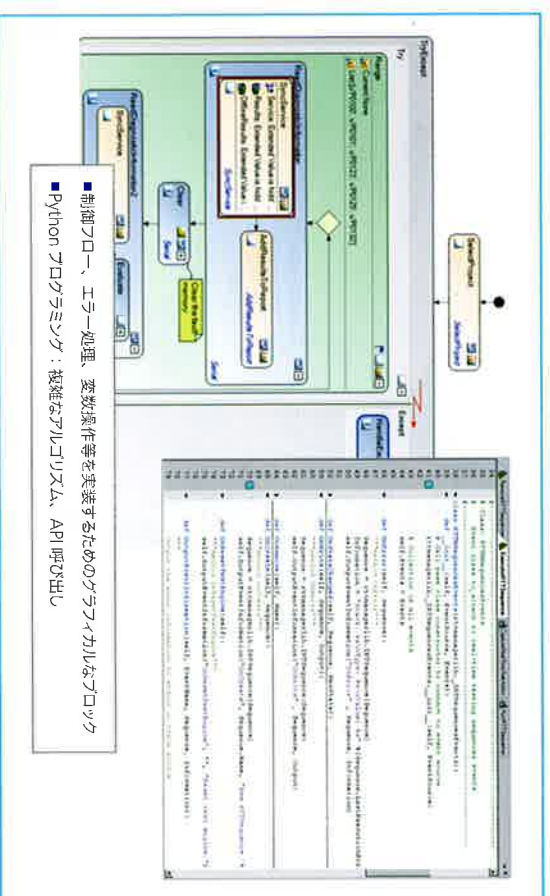
コントローラとHILシミュレータの接続においても、経験豊富なdSPACEエンジニアがハードウェアとモジュールのセッティングを行い、納品後にすぐHILシミュレーションを開始可能なターンキーソリューションをご提供いたします。

## 自動テスト環境

HILシミュレーションは制御対象がシミュレーションで実現されるため、テスト条件や繰り返し回数などをあらかじめ決めておき、自動でテストをすることが可能です。SPACEのHILシミュレータは自動テストを設定し、実行するための環境であるAutomationDeskによってテスト自動化を効率よく行うことができます。

HIL シミュレータはコントローラ以外をすべて  
 シミュレーションだけでなく、制御対象の  
 実物と組み合わせた検証も可能です。例えばコ  
 ントローラに実物のモーターを接続し、HIL シ  
 ミュレータで計算した軸負荷を利用してダイナ

ミクスをコントロールするシミュレーションベンチ  
 システムなどがあります。(軸負荷を発生する  
 モーターが検証するモーターに対して向かい合  
 わせとなります)



テスト自動化環境



シミュレーションベンチの例