ETAS

DRIVING EMBEDDED EXCELLENCE

ETAS INTECRIO V5.0

ユーザーガイド

www.etas.com

著作権について

INTECRIO V5.0 ユーザーガイド R03 JP – 11.2022

1 安全と個人情報保護についての注意事項

この章では、本製品の使用目的、対象ユーザー、安全上の注意、個人情報保護について説明しています。

本製品の使用にあたっては、ETAS Safety Advice(**Help → Safety Advice**) とユーザードキュメントに記載されている安全上のご注意に従ってください。

1.1 本製品に関する特殊な注意事項

本製品を安全に使用するために、以下の特殊な要件も守ってください。

- 本製品の準備や操作を行う前に、本製品を使用する環境が所定の条件を満たしていることを確認してください。各条件については、使用する PC やハードウェアのドキュメントを参照してください。

1.2 本書の対象ユーザー

本書は、組み込み制御システムの機能領域およびソフトウェア開発を専門とする、訓練を受けた従業員向けの内容となっています。

INTECRIO を使用するユーザーは、Windows® 8、Windows® 8.1、Windows® 10 の操作方法を理解している必要があります。メニューコマンドの実行やボタン操作の方法に加え、Windows のファイルシステム、特にファイルやディレクトリで構成される階層構造についての知識が必要です。また Windows ファイルマネージャや Windows エクスプローラ等の使い方、さらに「ドラッグアンドドロップ」操作に習熟していることも必要です。

Microsoft Windows のマウスおよびキーボード操作に慣れていない方や、一般的なファイルのナビゲーション方法についてご存知でない方は、まずそれらについて学習した後に INTECRIO をご使用いただくことをお勧めします。Windows オペレーティングシステムについての詳しい説明は、マイクロソフトのドキュメントを参照してください。

INTECRIO がサポートしている BMT（ASCET や MATLAB/Simulink）についての知識も必要条件となりますが、さらに、プログラミング言語、特に ANSI-C の知識があれば、INTECRIO を一層効率的にご使用いただけます。

1.3 使用目的

INTECRIO は、車両組み込みコントロールシステムのプロトタイピング向けの統合プラットフォームです。ASCET、MATLAB、Simulink、C コードなどのさまざまなソースから、ECU 上にアプリケーションソフトウェアを統合できます。

バーチャルプロトタイピングは、開発時間を短縮するということから、非常に重要視されています。INTECRIO のバーチャルプロトタイピング能力を使用すると、複雑なプロトタイピングハードウェアを必要とせずに、システムモデルの分析が可能となります。

INTECRIO のラピッドプロトタイピング能力を使用すると、実際の状況下、また実車両において、制御、診断機能の確認や検証が可能です。プロトタイプは、ETAS ラピッドプロトタイピングハードウェアを介して、既存の ECU 車両ネットワークに統合することができます。ETK、XETK、FETK、XCP を介したバイパスアプリケーションにおいて、INTECRIO は、ラピッドプロトタイピングハードウェアがシミュレーションコントローラーとして使用され、ECU の新機能に用いるパラメーターの計算をします。

製品の不適切な使用や安全に関する注意事項に従わないことにより生じた一切の損害について、ETAS GmbH は責任を負いません。

1.4 安全に関する注意事項の記述書式

安全に関する注意事項は以下の書式で記述されます。これらの情報は必ずよくお読みいただき、指示に従ってください。

危険

記載事項を守らないと死亡または重傷のリスクが高い危険性について説明しています。

警告

記載事項を守らないと死亡または重傷のリスクを招く可能性のある危険性について説明しています。

注意

記載事項を守らないと軽～中程度の負傷のリスクを招く可能性のある危険性について説明しています。

ご注意ください！

記載事項を守らないと物的損害を招く可能性のある状況について説明しています。

1.5 安全に関する情報

ユーザーの怪我や所有物の損傷を回避するため、安全に関する注意事項（「ETAS Safety Advice - 安全上のご注意」）、および下記の注意事項をよくお読みいただき、その指示に従ってください。

警告

不適切に初期化された NVRAM 変数は、車両やテストベンチの予期しない挙動を招く恐れがあり、そのような挙動により人身事故や物的損害が発生する危険性があります。

E-Target（実験ターゲット）の NVRAM 機能を使用する INTECRIO システムにおいては、ユーザー定義された初期化プロセス内で、すべての NV 変数の値がカレントプロジェクトに対して有効な状態になっているかを、個々の NV 変数単位および他の NV 変数との関連性においてチェックする必要があります。もし有効な状態になっていない場合は、すべての NV 変数をデフォルト値（有意で安全な値）に設定してください。

データ保存に関する NVRAM の特性から、不適切な初期値が使用されることにより人体や装置が傷付けられる可能性のある環境内（車上やテストベンチなど）においてプロジェクトが使用される場合は、特にこの注意事項を厳守してください。

ETAS 製品を安全にご使用いただくための注意事項（「ETAS Safety Advice - 安全上の注意事項」）は、以下のような形式でも提供されています。

- インストールパッケージに含まれる以下のファイル
  `Documentation\General\ETAS Safety Advice.pdf`
- 製品プログラム起動時、または **Help > Safety Advice** 選択時に表示される "ETAS Safety Advice" ウィンドウ

1.6 個人情報保護に関する注意事項

ユーザーの個人情報保護の問題は ETAS にとっても重要な案件であるため、本項では、INTECRIO 内でどのようなデータが処理されるか、どのような種類のデータを INTECRIO が使用するか、また個人情報の保護のためにユーザー自身がどのような技術的対策を講じるべきか、といった内容について説明します。さらに、本製品が個人データを保存する場所や、それらのデータの削除方法についても説明します。

1.6.1 データの処理

本製品の使用時には、個人データが処理されます。本製品の購入者は、GDPR（General Data Protection Regulation：EU の一般データ保護規則）の Article 4 No. 7 に従ってデータ処理を行う法的責任があります。製造者である ETAS GmbH は、当該データの不適切な扱いに関して、いかなる場合も責任を負いません。

1.6.2 データとデータカテゴリ

本製品は、エラー解析や外部プログラムとのやり取りなどの目的で、ソースライブラリを参照するファイル名とパスを含むファイルを作成します。

ファイルの名前とパスがカレントユーザーの個人ディレクトリやそのサブディレクトリを参照している場合は、以下のように個人データがファイル名とパスに含まれる可能性があります。

`C:\Users\<UserId>\Documents\...`

さらに、試験車両内で ETAS ラピッドプロトタイピングソリューションを用いて実センサや車載バス、ECU などに接続すると、ETAS ツールは運転者の個人データにアクセスすることが可能になります。

これらのデータは、INCA-EIP や ETAS Experiment Environment（ETAS 実験環境）によりデータロガーに保存される可能性もあります。

また、ETAS ライセンスマネージャでユーザーベースライセンスを扱うと、以下のような個人データが、ライセンス管理の目的で記録される可能性があります。

- 通信データ：IP アドレス
- ユーザーデータ：ユーザー ID、Windows のユーザー ID

1.6.3 技術的／組織的な対策

本製品は、個人データを記録する際に暗号化を行いません。記録されるデータの機密保持のため、ユーザー側の IT システムに適した技術的対策または組織的対策を講じてください。

ログファイル内の個人データは、オペレーティングシステムのツールを用いて削除することができます。

2 INTECRIO について

INTECRIO は、各種 BMT で生成されたコードを統合し、ラピッドプロトタイピング用にモデルプロトタイプやハードウェアの構成をセットアップして、実行コードを生成します。

本書は、INTECRIO 製品の概要説明とともに、迅速かつ確実な結果を得るための情報を提供するものです。本製品を使用して早く成果を出したいユーザーをサポートします。目的の情報を探しやすくし、段階的にシステムを紹介しています。

注記

入門ガイドの「INTECRIO について」も併せて参照してください。

本書は、ユーザーが INTECRIO の使用に習熟することをサポートします。

- **第 3 章 「INTECRIO を理解する」**
  INTECRIO の使用における重要なコンセプトについて説明しています。
- **第 4 章 「INTECRIO と AUTOSAR」**
  INTECRIO の AUTOSAR 対応機能についての概説です。
- **第 5 章 「INTECRIO のコンポーネント」**
  INTECRIO の各コンポーネントとその機能について説明しています。具体的な操作方法はオンラインヘルプを参照してください。
- **第 6 章 「SCOOP と SCOOP-IX」**
  SCOOP というインターフェース記述の概念と、その記述言語である SCOOP-IX について説明しています。
- **第 7 章 「モデリングのヒント」**
  INTECRIO におけるモデリングの概念を説明し、INTECRIO でラピッドプロトタイピングを行うためには BMT でどのようなモデリングを行えばよいかを、具体的に紹介します。
- **第 8 章 「バイパスの概念」**
  「フックベースバイパス」（“hook-based bypass”）、「サービスベースバイパス」（“service-based bypass”）の概念に関して説明しています。
- **第 10 章 「用語集」**
  重要な略語と用語についての説明です。

3 INTECRIO を理解する

昨今の ECU 開発においては、ターゲットハードウェアが存在しない状態で組み込み制御ソフトウェアの制御アルゴリズムを開発しなければならない、というケースが多くなっています。この際、アルゴリズムは ASCET や MATLAB®/Simulink® などの BMT、つまり、記述されたモデルからソフトウェアコードを生成できるツールを用いて作成されます。そして、ターゲットハードウェアに代わるものとして、ETAS の仮想プロトタイピングシステムやラピットプロトタイピングハードウェアシステム（ES900、ES800 など）が使用されます。

仮想プロトタイピング（「バーチャルプロトタイピング」とも呼ばれます）やラピッドプロトタイピングを行うための統合ソフトウェアプラットフォームである INTECRIO は、組み込み制御ソフトウェアの開発に携わるエンジニアの日々の作業を支援する ETAS 製品ファミリの 1 つです。

INTECRIO は、各種 BMT で生成されたコードを統合し、ラピッドプロトタイピング用にモデルプロトタイプやハードウェアの構成をセットアップして、実行コードを生成します。そして ETAS Experiment Environment（ETAS 実験環境）を利用して仮想プロトタイピングまたはラピッドプロトタイピングの実験をリアルタイム条件下で実行し、値の測定／適合などを行うことができます。

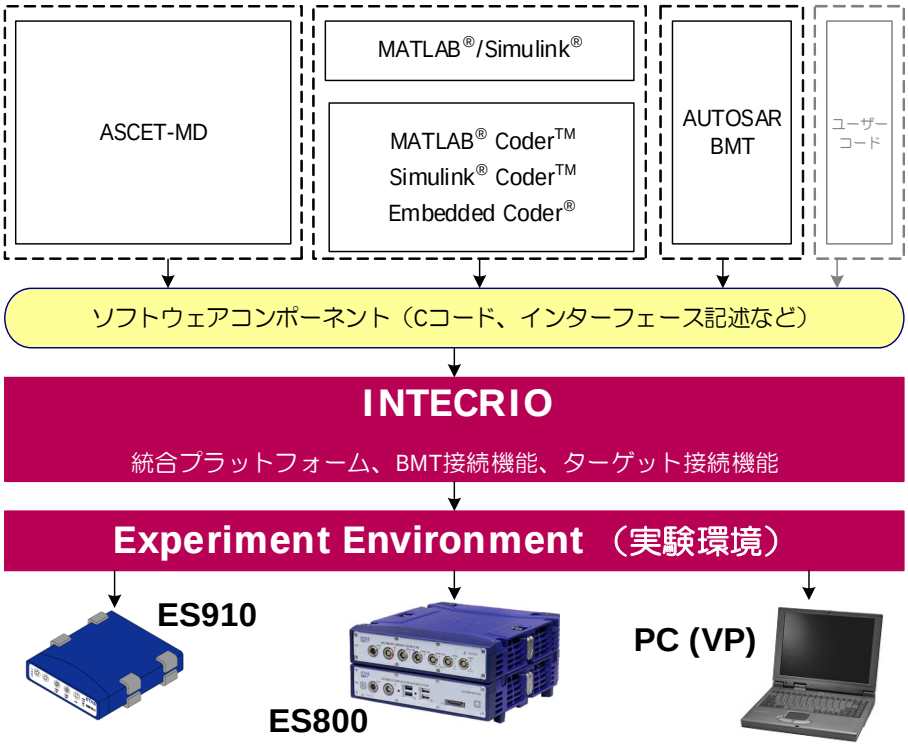


図 3-1 INTECRIO — 概要

INTECRIO について詳しく説明する前に、ここではまず、ECU 開発全般、仮想プロトタイピング、開発工程における INTECRIO の位置付け、といった、INTECRIO を使用する際の背景や課題について、要点をまとめます。

3.1 ECU 開発における課題

今日、ECU 開発のプロセスは非常に複雑であるといわれています。その主な理由としては、ハードウェアとソフトウェアの要件が高度化され、また開発作業が複数のメーカーやサプライヤに分散されることが多い、といった点があげられます。

3.1.1 システム要件の複雑性

一般的な ECU ソフトウェアは、ECU の制御アルゴリズムを実行するアプリケーションコード部分と、ハードウェア用コード部分とで構成されていますが、通常、ECU ソフトウェアは実行されるターゲットシステムに密接に依存しています。

ETAS では、このような相互依存を解消する第一歩として、ハードウェアに依存しないソフトウェア（制御アルゴリズム）を記述する手法を、ASCET で実現しました。ASCET は、制御アルゴリズムから得られる「シグナル」をハードウェアに接続される「シグナル」にマッピングすることにより、これを実現しています。ラピッドプロトタイピングシステムをターゲットとして用いる場合、ハードウェアの構成と設定は専用エディタで記述します。また、各種マイクロコントローラターゲット用のソフトウェアを生成するためのオプション機能も用意されています。

将来の ECU 開発を成功させるためには、ソフトウェアとターゲットハードウェアをいかにスムーズに切り離して扱えるようにするかが大きな課題であるといえます。

この「切り離し」がスムーズに行われれば、ECU ソフトウェアとハードウェアを並行して開発することが可能となるため（図 3-2 参照。この図は V モデルをベースにしています）、開発時間の短縮にも繋がります。

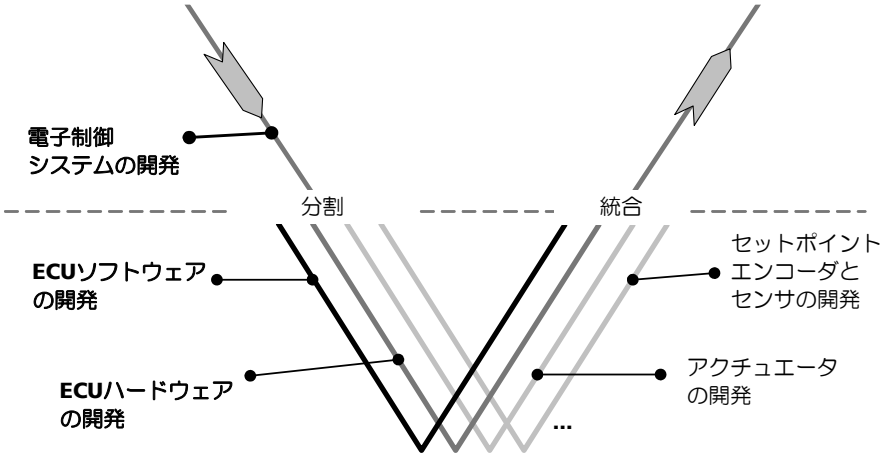


図 3-2 電子システム開発の概要

しかし一方で、最近はシステムそのものがますます複雑になり、複数の ECU 間の相互関係も重要になってきています。自動車 1 台当たりの ECU 数と ECU1 基当たりのファンクション数は、この数十年間増加の一途をたどっています（図 3-3 参照）。

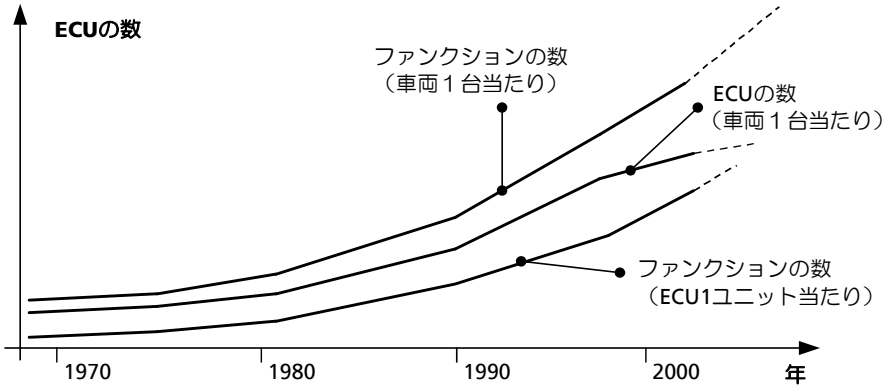


図 3-3 自動車 1 台当たりのファンクションと ECU の数（ミュンヘンで開催された第 2 回ヨーロッパシステムエンジニアリング会議における H. G. Frischkorn、H. Negele、J. Meisenzahl によるキーノート、*The Need for Systems Engineering. An Automotive Project Perspective* より）

自動車メーカーとサプライヤにとってコスト削減は大きな課題であるため、既存のリソースの再利用やバリアント活用の重要性が非常に高くなっています。このため、たとえば自動車メーカーにおいては、ある 1 つの ECU ファンクションのソフトウェアプロパティを変更して車両バリアントを作成し、各バリアントを異なる複数の車両タイプに流用する、といった方策がとられています。極端な例では、ある車種に 2 種類のタイプがあった場合、それらの違いが、ある特定のソフトウェア機能とそれに関連するセンサ／アクチュエータの有無だけである、というケースもあります。

このような背景から、ハードウェアに依存しない ECU ファンクションを開発する、ということは、そのファンクションを幅広く活用できるという大きなメリットにつながります。

また、小規模のハードウェアシステム上にソフトウェアコンポーネントを配置することも、コスト削減に役立ちます。しかしこの場合、1 つのソフトウェアコンポーネントを複数の ECU に分散させる必要が生じます。

さらに、「仮想センサ」と呼ばれるコンセプトを用いることにより、センサを追加することなく、車両に新機能を追加することができます。仮想センサは、実際の信号測定を行わず、演算によりセンサシグナルを生成します。この演算は、物理モデルと既存のセンサ値とを組み合わせて実現します。この典型的な例が、タイヤスリップの測定です。タイヤスリップ自体を測定することはできませんが、タイヤモデルと、ESP（**E**lectronic **S**tability **P**rogram）で測定した現在の加速度、さらにエンジン制御システムから得られる現在のトルクを組み合わせることにより、現在のスリップの度合いを演算により求めることができます。

今日の自動車に搭載されたマイクロコントローラの数は、最大で 120 基にも及び、それらは通常、シリアルバスシステムで相互に接続されています。そのため ECU 上で実行される制御アルゴリズムは、その ECU 自体だけでなく他の ECU の入出力にも依存することになります。

分散された制御アルゴリズムの例としては、ESP やエンジン制御があります。

ESP システムは、危機的な運転状況を検知すると直ちにエンジン制御に対してトルクダウンを要求します。一般的に、この要求は CAN バス経由で転送されます。大抵の場合、トルクダウンによって車は直ちに安定し、再びトルクを増加できるようになりますが、安定しない場合は、更なるトルクダウンが必要となります。

この制御アルゴリズムには以下のコンポーネントが関与しています。ESP はトルクダウンの必要性を検知して CAN メッセージを送信します。この際、CAN バスの空き時間を待つ必要があり、CAN バスの負荷状態と通信設定によっては、多少時間がかかる場合があります。次に、メッセージを受け取ったエンジン制御ユニットが、要求に応じてトルクダウン処理を行います。エンジン制御ユニットは、常にタイムクリティカルなタスク（つまり通常のエンジン制御）を実行しているため、適切な時間枠が見つかった時点で直ちにトルクダウンを実行します。しかしこの際、処理待ち時間が非常に長くなってしまえば、車両の挙動は運転者にも感じられるほど不完全なものとなってしまうため、このような状況は避けなければなりません。

この例からも、自動車の電子技術が複雑化していることがよくわかります。今日の自動車の技術革新は、その 90%が電子技術をベースとしたものであると言われており、電子技術は自動車業界での成功に重要な役割を果たしています。そのため、自動車メーカーとサプライヤは、一様に自動車の電子技術の向上に非常に強い関心を抱いています。

3.1.2 分散開発により生じる複雑性

ECU 開発を複雑にしている要因のひとつとして、一連の開発が自動車メーカーとサプライヤの分業体制によって行われていることがあげられます。一般的に、自動車の制御ファンクションについての要件を自動車メーカーが定義すると、そのファンクションを電子システムとして実装する作業は ECU サプライヤが行います。そして、実装されたファンクションの実車での調整や受け入れ検査は、自動車メーカーの責任において行われます。

このような分業体制での開発を成功させるためには、自動車メーカーとサプライヤの間の「インターフェース」（ここでの「インターフェース」は、「責任範囲の境界線」を指します）を明確に定義することが不可欠です。このインターフェースは、図 3-4 のような V モデルで表現できます。車両は、V モデルの左右の大きな枝に相当し、自動車メーカーがこれについて責任を持ちます。そして多くの場合、サプライヤが各コンポーネントレベルを担当し、さらには統合ステップの最初の段階を担当する場合もあります。

この際のインターフェースの定義は、場合により異なることもありますが、個々のインターフェースは正確かつ完全に定義されていなければなりません。

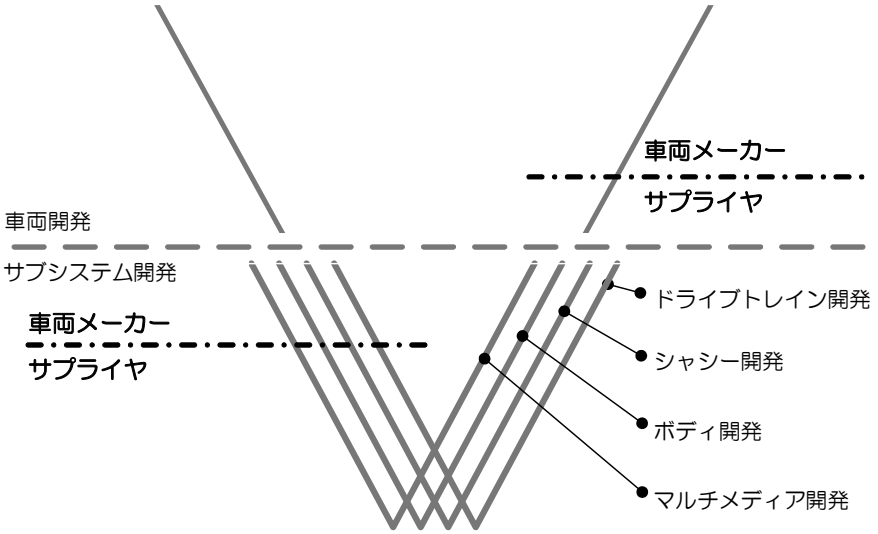


図 3-4 自動車メーカーとサプライヤの責任範囲。インターフェースが必要となる箇所（一例）を鎖線で示しています。

近年、欧州の自動車メーカーとサプライヤとで組織された「AUTOSAR」により、基本システムとインターフェース機能の標準化が図られていますが、INTECRIO V5.0 はこの「AUTOSAR 仕様」をサポートしています。詳しくは第 4 章を参照してください。

3.1.3 新しい開発手順

自動車に新しい機能を追加する場合、コスト削減の最良の方法は、まず最初に機能を定義し、次にその機能を実現するために必要な最低限のハードウェアシステムを構築することである、とも言われています。

しかしこれでは自動車のハードウェアをすべて開発しなおすことになってしまうので、膨大な労力を伴い、リスクも増大します。そこで、今日の自動車開発においては、電子システムに機能を実装し、その内容を新しい機能に合わせて調整していく、という方法がより有効です。INTECRIO が開発されたのはこのような理由によるものです。機能が制御可能な状態になれば、すぐに次の適合工程に進むことができます。

詳しい情報は、以下の書籍に記載されています。

- J. Schäuffele、Dr. T. Zurawka：“Automotive Software Engineering - Principles, Processes, Methods, and Tools”

3.2 電子システムの定義

ECU に組み込むソフトウェアを作成するには、ハードウェア（ドライバなど）とソフトウェア（制御アルゴリズムを実行するアプリケーション部分）の両方を定義する必要があります。前述のように、今日の制御ソフトウェアは、単なる「独立した ECU 上で稼働するソフトウェアコンポーネントの集合体」として考えることはできません。多くの場合、1 つの制御アルゴリズムは複数の ECU で分散制御されるため、それら全体を把握して定義する必要があります。

ただし INTECRIO V5.0 ではまだネットワーク機能はサポートされていないため、本項では個々の ECU ソフトウェアの開発に焦点を当てて説明します。

3.2.1 電子システムの設計と動作方法

本項では、自動車の電子システムの設計と運用について、電子油圧式ブレーキシステムを例に詳しく説明します。

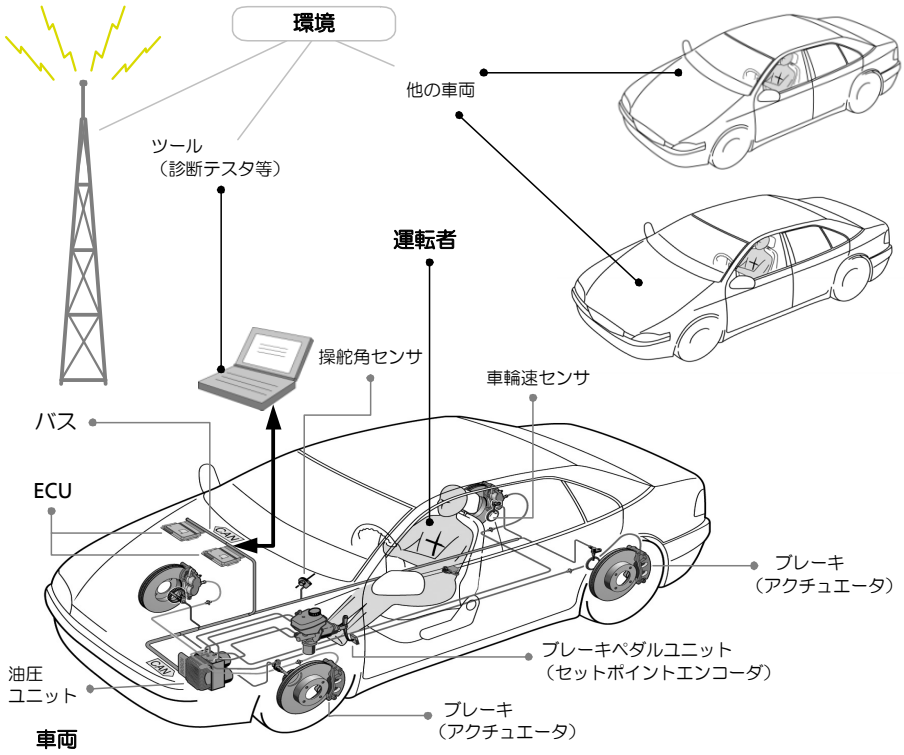


図 3-5 電子油圧式ブレーキの設計（出典：Robert Bosch GmbH(ed.), Stuttgart, 2002 年出版の *Konventionelle und elektronische Bremssysteme*）

図 3-5 は、Bosch 社の電子油圧式ブレーキシステム - SBC（**S**ensotronic **B**rake **C**ontrol：センソトロニックブレーキコントロール）の設計の概要を示しています。SBC は、パワーブレーキユニット、アンチロックブレーキングシステム（ABS）、および ESP の機能を一体化したものです。

ブレーキペダルが運転者によって機械的に踏まれると、それがブレーキペダルユニット内で感知され、電気信号として ECU に伝えられます。電気信号を読み込んだ ECU は、さまざまな内部値やセンサ信号（ステアリング角信号や車輪速信号など）を使用して出力量を計算します。その出力量は、電気信号として油圧ユニットに伝えられ、圧力変調により車輪ブレーキ用の被制御変数に変換されます。そして車輪ブレーキ経由で、被制御システムである「車両の走行」が影響を受けます。このため、この制御系において車輪ブレーキは「*アクチュエータ*」とも見なされます。

ECU は、同じ車両内の他の ECU と、CAN などのバス経由で通信することができます。

図 3-6 では、SBC のシステム設計の例を紹介していますが、これは、車両のすべての電子制御（閉ループ制御）や監視システムに広く用いられる典型的な設計様式です。ここでは、車両のコンポーネントとして、*セットポイントエンコーダ*、*センサ*、アクチュエータ、各種 ECU などがあげられます。また ECU をネットワークに接続してデータを交換することも可能です。

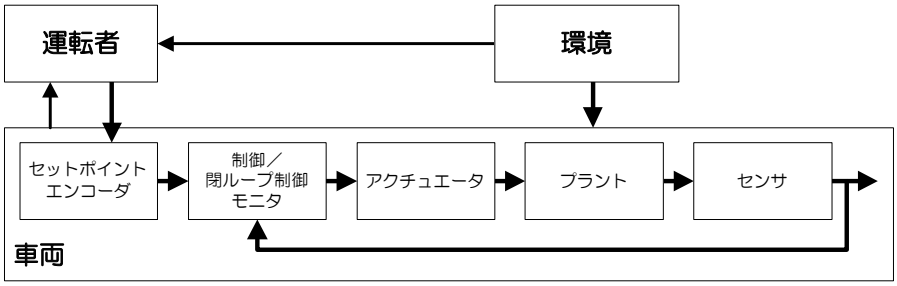


図 3-6 閉ループ制御と監視システムの略図

運転者（乗客も含まれる場合があります）と環境（他の車両や、車両に搭載された診断ツールのような電子システムも含みます）は車両の挙動に影響を与える可能性があり、これらは、「車両 - 運転者 - 環境」というさらに高いレベルのシステムのコンポーネントとなります。

これらのことからわかるように、ECU 自体は一つの「手段」に過ぎません。ECU がセットポイントエンコーダ、センサ、アクチュエータと連携して完全な電子システムを構成することによって初めて、車両走行に影響を与え、さらにそれを監視することが可能となり、利用者の期待に応えることができます。

3.2.2 電子システムのアーキテクチャと記述

電子システムのソフトウェアを記述するには、ECU と車両のネットワーキング、および既存ハードウェアへのソフトウェアの分散を定義する「物理アーキテクチャ」が必要です。この物理アーキテクチャには、ECU の設置スペースや、安全用コンポーネントに必要な冗長性などの要件も含まれます。

一方、ソフトウェアの機能を定義する「論理アーキテクチャ」も必要です。この理想的なものは、システム全体のレイアウトから始まり、ファンクション、モジュール、さらに個々のクラスに細分されるものです。

モジュールやクラスと同等に扱われるものとして、「AUTOSAR ソフトウェアコンポーネント」（SWC: AUSTOSAR **S**oft**w**are **C**omponent）も使用可能です。

またこれらのアーキテクチャ（物理アーキテクチャと論理アーキテクチャ）を結び付け、1 つのファンクションをハードウェアに分散させることも不可欠です。

図 3-7 は、これら両方のアーキテクチャを示しています。INTECRIO は赤い線で囲まれた部分の開発に利用できます。

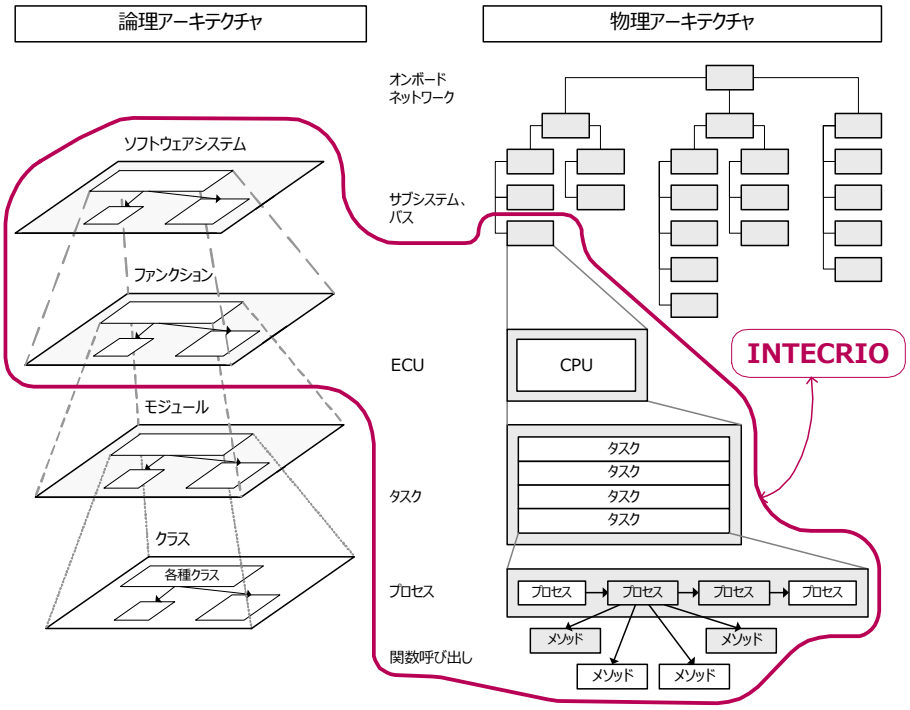


図 3-7　ECU ソフトウェアのアーキテクチャ

図 3-8 に、INTECRIO の観点から見た「ECU のソフトウェア記述」について、詳しく説明します。図 3-8 は、ECU ソフトウェアに含まれる各種コンポーネントの概要を示しています。

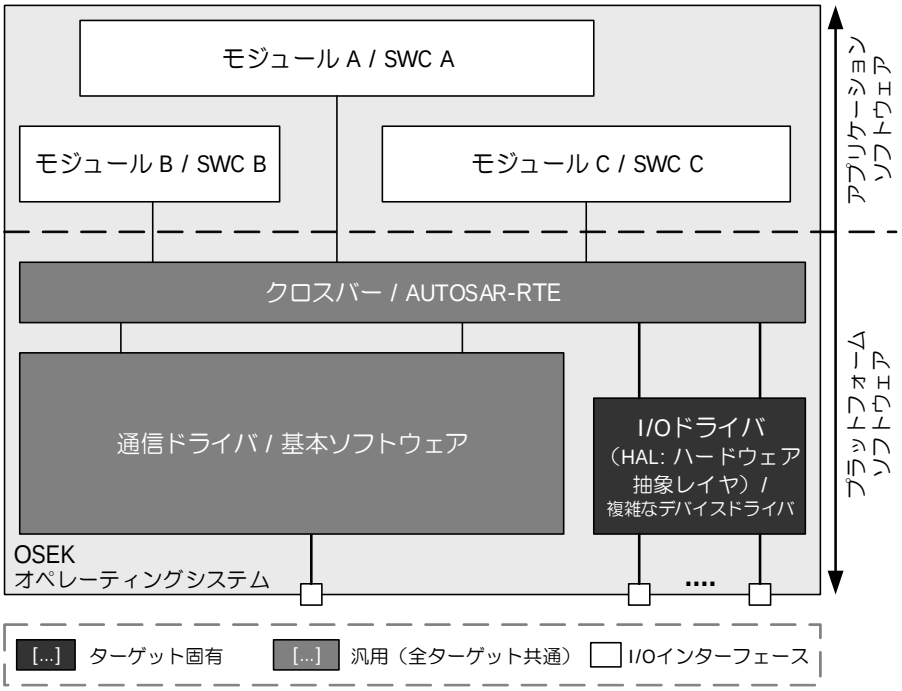


図 3-8　ECU ソフトウェアの内部構成：INTECRIO の観点から見た概略

3.2.2.1 アプリケーションソフトウェア

「アプリケーションソフトウェア」（または「ファンクションソフトウェア」とも呼ばれます）は、シグナルフローに基づく制御アルゴリズムが要件と仕様に基づいて汎用的に記述されたものです。このソフトウェアは、個々の「モジュール」／「SWC」（AUTOSAR ソフトウェアコンポーネント）と、ファンクションを表すモジュール／ SWC のグループで構成されます（図 3-9、および図 3-7 の左側部分を参照してください）。

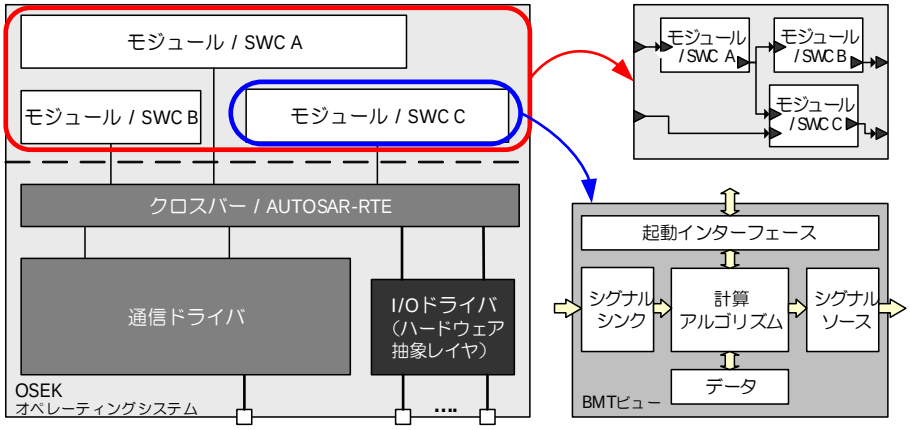


図 3-9 ファンクションソフトウェア：詳細

図 3-2 のシステム開発の概要図では、ECU ソフトウェア開発の全体が 1 つの開発フェーズとして示されていますが、3.1.2 項の図 3-4 ではこれが複数のフェーズに分割されています。しかしそれでもまだ実際の開発フェーズを表したものにはなっていません。1 つの機能を開発する際、それに含まれる複数のモジュールや SWC、ファンクションを、複数のサプライヤが異なるツールを用いて開発する可能性もあるからです。

INTECRIO においては、すべてのモジュールと AUTOSAR ソフトウェアコンポーネントは基本的にはどれも同じ構造で、以下のインターフェースを備えています。

- シグナルシンク — 入力、またはクライアントとレシーバ
- シグナルソース — 出力、またはサーバーとセンダ
- 起動インターフェース — プロセス、またはランナブルエンティティ（RE、実行可能なエンティティ）を指します。次の図には示されていません。

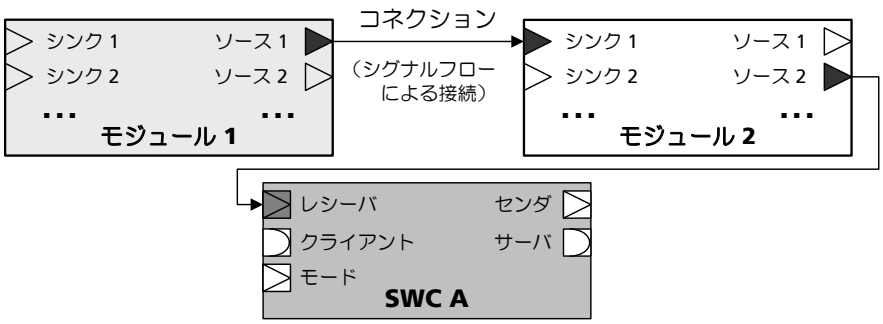


図 3-10 モジュール／ SWC：構成の概略（外観図）と接続

モジュールと SWC（AUTOSAR ソフトウェアコンポーネント）の内部構造も、すべて共通です。次の内部構成図には、各種インターフェースに加えて以下のコンポーネントも示されています。

- 計算アルゴリズム（機能記述）
- データ（変数、定数、パラメータ）

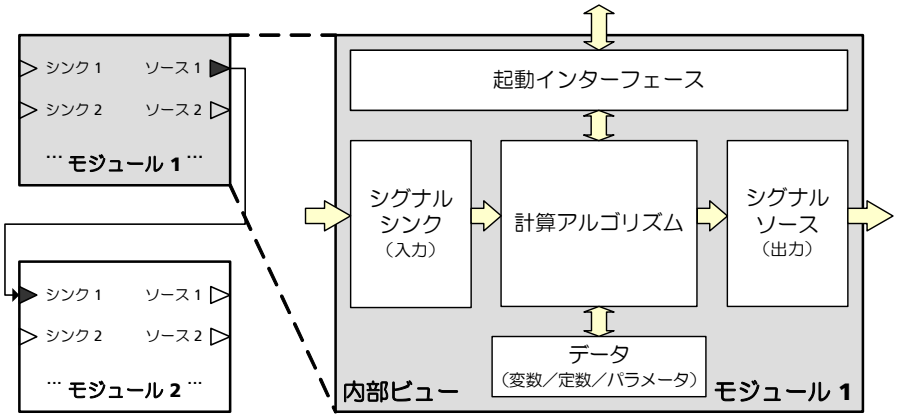


図 3-11 モジュール：内部構成図

図 3-12 に、シンプルな ASCET モジュールの例を紹介します。

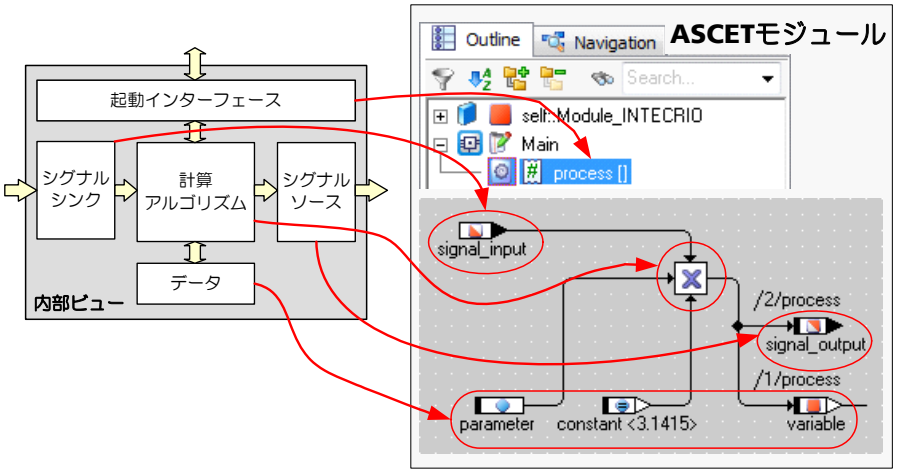


図 3-12 モジュール：ASCET モジュールの例

「起動インターフェース」は ASCET の「プロセス」に相当し、「シグナルシンク」と「シグナルソース」は ASCET の「受信メッセージ」と「送信メッセージ」に相当します。変数、パラメータ、定数は、それぞれ同名の ASCET オブジェクトで表現されます。

図 3-13 は MATLAB®/Simulink® の例です。

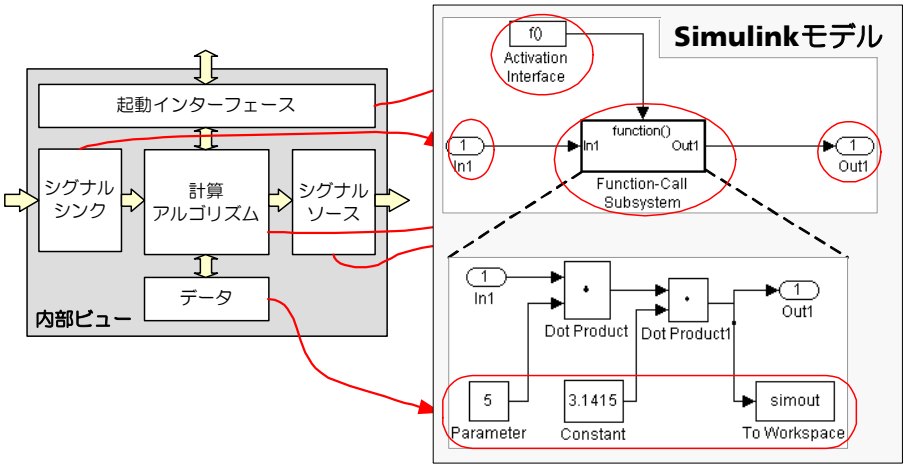


図 3-13　モジュール：Simulink® モジュールの例

モジュール／ SWC、またはファンクション間で情報が交換され、全体として機能するシステムを作成するためには、これらのオブジェクトを「接続」、つまり「統合」する必要があります。接続の際には、個々のモジュールの計算アルゴリズム、つまり個々のモジュール／ SWC の機能自体に関する情報は必要はないので、モジュールは以下の式のように「ブラックボックス」として扱われます。

$\mathtt{Source\ 1} = f_1(\mathtt{sink\ 1,\ sink\ 2,\ ...})$

$\mathtt{Source\ 2} = f_2(\mathtt{sink\ 1,\ sink\ 2,\ ...})$

この「統合」を実現することが、INTECRIO の役割です。統合には、任意の BMT で記述されたモデルの各部分（インターフェース、データ、C コード）をそれぞれ別のファイルに出力したものを使用します（図 3-14 参照）。これらのファイルによって再利用可能なソフトウェアコンポーネントが形成され、INTECRIO での「統合」が可能となります。

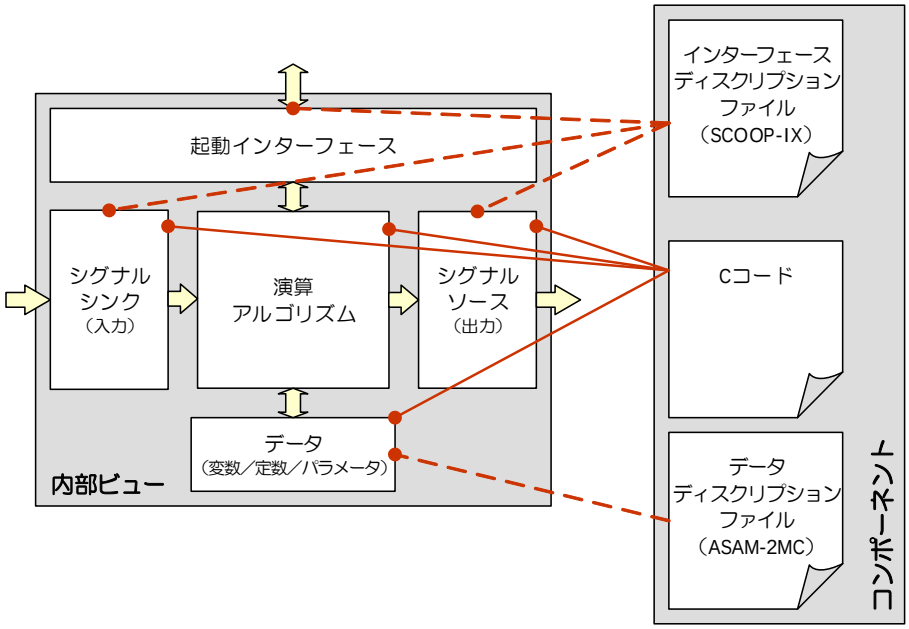


図 3-14　モジュールの内部構成図とコンポーネント図
（破線：記述、実線：実装）

3.2.2.2 プラットフォームソフトウェア：ハードウェアシステム

ECU ソフトウェアの中には、ハードウェアシステムについての記述部分があり、この部分は車両に応じて異なるため、コスト要因の一つであるといえます。ここでの「ハードウェア」とは、車両に搭載される ECU ネットワークを指します。ハードウェアシステムには、ソフトウェアによって使用されるハードウェアシステムの全側面が記述されていなければなりません。ハードウェアシステムも、挙動やプロパティが一貫した汎用システムとして記述されます。

そして仮想プロトタイピング（バーチャルプロトタイピング）においては、実際のハードウェアや環境の代わりに各種モデル（運転者、車両、環境）が使用されます。

3.2.2.3 ハードウェアとソフトウェアの接続

ハードウェアシステムとソフトウェアシステムは、互いに接続することによって機能します。プロトタイプ（または実際の車両プロジェクト）を構築するには、ソフトウェアシステムの記述とハードウェアシステムの記述を結び付ける必要があります。このような「汎用的なソフトウェア記述」と「汎用的なハードウェア記述」の接続を実現するには、特殊な「接着剤」が必要です。

この「接着剤」の役割を果たすのが、INTECRIO の *「プロジェクトコンフィギュレータ」* です。これは、BMT から提供されるアプリケーションソフトウェア用コンポーネントと、INTECRIO で設定されるハードウェア記述を接続するためのツールです。

図 3-15 は、ECU ソフトウェアの各コンポーネントと、INTECRIO の各コンポーネントとの関連を表しています。

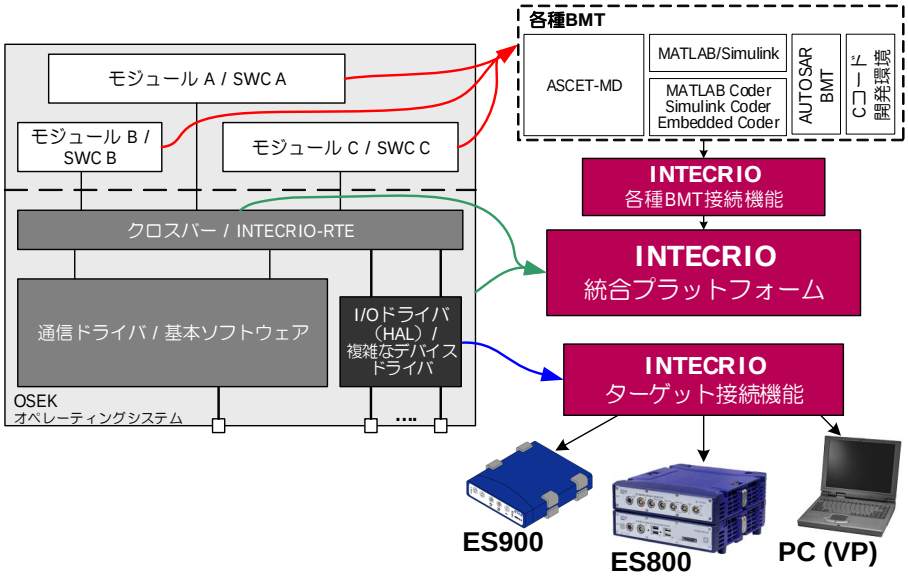


図 3-15 システムプロジェクト（ECU ソフトウェア）と INTECRIO コンポーネント

3.3 仮想プロトタイピング

「仮想プロトタイピング」（=「バーチャルプロトタイピング」）とは、ファンクション開発者が自動車電子システムの「仮想プロトタイプ」（実行ファイル）を作成して、それを PC 上でテストすることを意味します。「仮想プロトタイプ」は以下の要素で構成されます。

- 自動車用組み込みソフトウェア

  - アプリケーションソフトウェア（制御／監視ファンクション）
  - プラットフォームソフトウェア（I/O ドライバ、オペレーティングシステムなど）
- プラントモデル（制御対象モデル）
  - 運転者
  - 車両
  - 環境

仮想プロトタイピングにより、開発工程内の初期段階から、ファンクション開発者とシステム開発者、およびシミュレーション技術者の共同作業が可能となります。従来の開発技法では、HiL（Hardware-in-the-Loop）テストフェーズのような、かなり後の工程にならないとこれらのエンジニア間のコンタクトは実現しませんが、仮想プロトタイピングを導入すれば、ファンクション開発者は開発の初期段階においてシステムモデル（シャシーやエンジンなどのモデル）を利用できるので、開発対象のファンクションの評価、つまり妥当性の確認を Model-in-the-Loop（MiL）技術を用いて行うことができます。またソフトウェアシステム化のための情報が早期に得られるため、ファンクション開発とシステム開発の理想的な相乗効果が生まれ、開発作業の効率が向上します。

3.3.1 ターゲットに近い環境でのプロトタイピング

INTECRIO では、複数の BMT ツール（MATLAB®/Simulink®、ASCET、C コード）で作成したモデルを組み合わせて用いることができます。また INTECRIO-VP の PC 接続機能（PC Connectivity）を使用して標準の Windows PC を用いた「仮想プロトタイピング」を行うこともできます。これにより、開発工程の初期段階であるファンクション設計の段階において、プラントモデルを対象に、機能的アーキテクチャの評価と電子的アーキテクチャの検証が行えます。しかもこれらの作業はすべて実際のターゲットに近い条件下で行えます。これをわかりやすく式にまとめると、以下のようになります。

```
  INTECRIO 統合プラットフォーム
+ ファンクションモデル
+ プラントモデル
+ 標準 PC
+ PC 用 RTA-OSEK
_____________________________________________
= INTECRIO での仮想プロトタイピング
```

INTECRIO-VP は、PC 用の完全な OSEK オペレーティングシステム「RTA-OSEK for PC」をサポートし、実際の ECU 上でシステムが稼動する際の状況（タスク／プロセス指向のスケジューリングや OS プロセス間のバッファメッセージ通信など）を、ほぼそのまま実現します。さらに PC 上でのテストでは、柔軟な実行条件や、高速応答、といったメリットを活用できます。ターゲット ECU に比べてタイミングや所要メモリ等の制約が少ないため、余裕のある実験環境を実現できます。

3.3.2 仮想プロトタイピングの優位性

仮想プロトタイピングにより、ECU 開発の早い段階において、オフィスで事前適合を行ったり、ファンクション挙動の詳細な分析やプロトタイプの実行速度制御などを行う機会が生まれました。このことを、以下に 3 つの例で説明します。

A 事前適合の時間と費用の節約

仮想プロトタイピングを行うと、テストベンチで行っていた作業を実験室やオフィスのデスク上に移すことができ、そこで、プラントモデルを対象としたファンクションの評価（妥当性の確認）や最適化、さらには事前適合を行うことができます。PC の環境下では、実験の実行速度を高速化できるため（ただし CPU の演算能力やモデルの複雑性に応じてその度合いは変わります）、さまざまなファンクションやデータのバリエーションを、より短い時間でテストすることができます（図 3-16 の下部参照）。

B. 複雑なシミュレーションモデルを用いる詳細分析

ファンクションの挙動を詳細に分析することができ、高度かつ複雑なプラントシミュレーションを対象とした新機能の評価、つまり妥当性の確認を実現できます。これは、実際には実現不可能な特殊な環境下においての詳細な評価が必要な場合に、特に有効です。

C. スローモーションと早送り

INTECRIO-VP を使うと、シミュレーション実行中に時間スケールを変更し、シミュレーション時間の速さを変化させることができます。スケーリングファクタを 1 未満に設定するとシミュレーション速度が速くなり、1 より大きくするとスローモーションで実行できます（図 3-16 の上部参照）。これにより、シミュレーション実行時に、特に詳しく調べたい部分をスローモーションで実行させることができます。

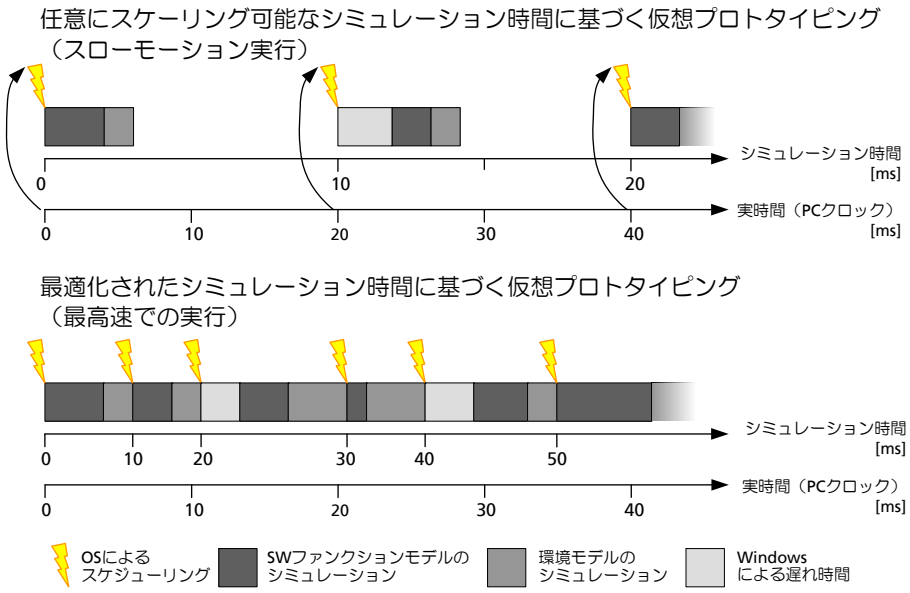


図 3-16 上：時間のスケーリングを指定し、（CPU の演算能力とモデルの複雑性による制約範囲内で）モデルをスローモーションまたは早送りで実行
下：理想化された実行制御により、モデルを最高速で実行

3.3.3 仮想プロトタイピングとラピッドプロトタイピング

INTECRIO の環境においてファンクションモデルは、OS コンフィギュレーションやハードウェアコンフィギュレーション、さらにはインストゥルメンテーションと厳密に切り離されて扱われるため、必要なモデルバリアントは最小限に留めることができます。また、複数の開発チームや、ターゲットプラットフォーム、さらに複数の開発段階において、ソフトウェアプロトタイプ、実験、データセットなどを共有できます。仮想プロトタイピングモデルをラピッドプロトタイピングに再利用することもできるので、これらの相乗効果も期待できます。

仮想プロトタイピングとラピッドプロトタイピングのメリットをまとめると、次の表のようになります。

仮想プロトタイピング	ラピッドプロトタイピング
ファンクションとソフトウェアの評価を行える	ファンクションとソフトウェアの検証（テスト）と評価（妥当性確認）が可能
• PC 上でシミュレートされた世界でモデルを実行 • 事前適合が可能 • 専用ハードウェアは不要 • リアルタイム要件の制約を受けない	• 専用ハードウェアシステム上でモデルを実行 • 実際のインターフェースと入出力信号を用いてモデルを実行 • バイパス実験などによるリアルタイム実行

このように、仮想プロトタイピングは、INTECRIO の従来のプロトタイピング技法を補完するものです。INTECRIO-VP を用いることにより、新しく実装する機能のテストをより早い段階においてプラントシミュレーションに対して行うことができるため、開発プロセスの効率がさらに向上します。

3.4 開発プロセスにおける INTECRIO の役割

一般に、ECU ソフトウェア開発は V モデルに従って進められます。そして V モデル内の各過程においても、さらに小規模な V サイクルの工程が採用されます。図 3-17 はこの V モデルの概略図で、INTECRIO が使用される工程が明記されています。

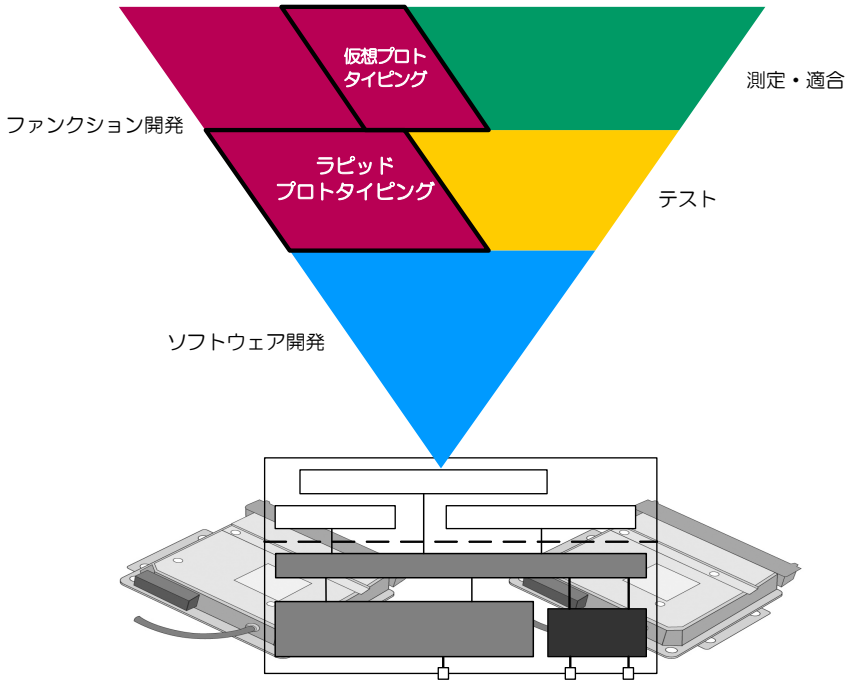


図 3-17 V サイクルと INTECRIO

INTECRIO は、複数の開発者がさまざまな BMT を用いて作成したソフトウェアコンポーネントを手早く統合し、統合されたソフトウェアの全体または一部の「検証」と「評価」を、仮想／ラピッドプロトタイピング手法を用いて行うためのツールです。

「Validation：評価」（または「妥当性確認」）とは、システムまたはコンポーネントが、アプリケーションの目的またはユーザーの要求に対応できているかどうかを判断することを指します。つまり評価の工程においては、ソフトウェア仕様が要件に対応しているか、そして最終的に顧客の満足が得られるものであるかどうかを確認します。

「Verification：検証」とは、システムまたはコンポーネントの動作をテストし、各開発フェーズでの成果物がそのフェーズの仕様を満たしているかを判断するものです。つまりソフトウェア検証の工程においては、現在の開発フェーズ用に定義された仕様が、ソフトウェアに正しく実装されているかどうかを確認します。

従来の方法でソフトウェアの開発や統合、品質保証を行っている場合、評価と検証を明確に区別することは容易ではありません。そのため、INTECRIO をラピッドプロトタイピングツールとして使用することの大きなメリットは、開発初期の段階で、ECU に依存しない実験環境においてファンクションの評価を行える、ということにあります。

3.5 INTECRIO の作業環境

INTECRIO は、モジュール式に設計されています。メインウィンドウである「グラフィカルフレームワーク」内に、INTECRIO のさまざまなコンポーネントが統合されています。

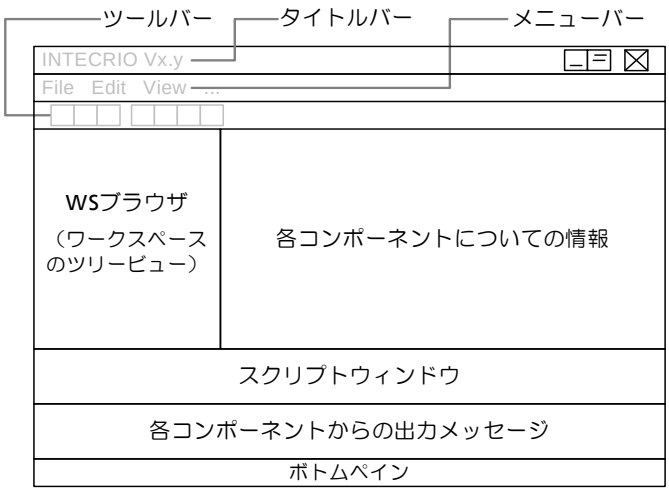


図 3-18 INTECRIO のユーザーインターフェース構成

メニューバーとツールバーの下の左側部分には、ワークスペースの内容がツリー表示されるワークスペースブラウザウィンドウがあり、その右側のフィールドに、現在使用されているコンポーネント（OS コンフィギュレータ、プロジェクトコンフィギュレータ、ハードウェアコンフィギュレータ）のユーザーインターフェースが表示されます。その下には、スクリプティングウィンドウ（詳細はオンラインヘルプを参照してください）と、現在使用されているコンポーネント用のメッセージウィンドウがあります。最下部のボトムペインには、各種ステータス情報が表示されます。各ウィンドウは、通常は図 3-18 のようにフレームワークにドッキングされていますが、必要に応じてフレームワークから切り離すこともできます。

左上の WS ブラウザウィンドウには、現在のワークスペースの内容がツリー構造で表示されます。ワークスペースでは ECU ソフトウェアを定義するためのさまざまなコンポーネントが管理されます。各コンポーネントは、ハードウェアシステム、ソフトウェアシステム、環境システム（仮想プロトタイピング用）、シス

テムプロジェクトに分類され、それぞれプリセットされた 4 つのメインフォルダ（Software、Hardware、Environment、 System）に格納されます。このワークスペース内で ECU ソフトウェアの定義を行います。

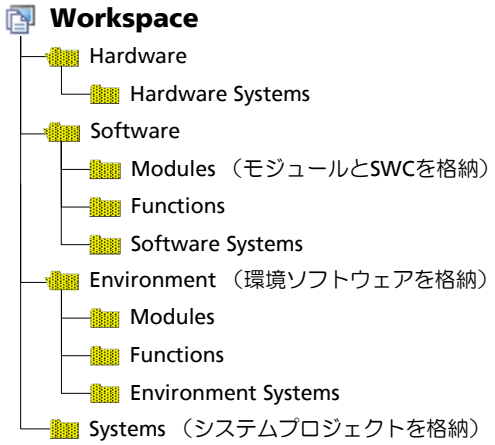


図 3-19 フォルダ構造：ワークスペース

ハードウェアシステム

「ハードウェアシステム」は、実験用ハードウェアの全体を定義するコンポーネントで、この情報をもとに、プロトタイプ内のプラットフォームソフトウェアに含まれるハードウェア記述部分が生成されます。1 つのハードウェアシステムには、ECU（ラピッドプロトタイピング／仮想プロトタイピング用実験ターゲット）についての定義と、デバイス間インターフェース（バスシステム）の定義が含まれます。

ハードウェアシステムの各コンポーネントは、**Hardware** フォルダに格納されます。

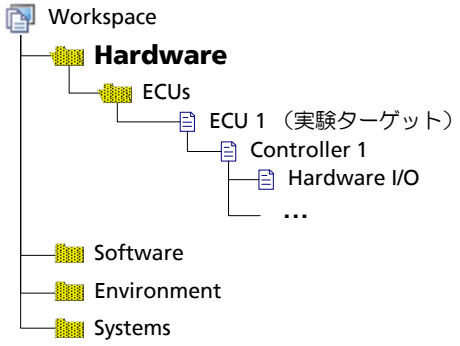


図 3-20 フォルダ構造：ハードウェアシステム

ソフトウェアシステムと環境システム

「ソフトウェアシステム」（Software System）は、アプリケーションソフトウェア、つまり制御アルゴリズムの汎用部分全体を定義するものです。ソフトウェアシステムには、機能が記述されたモジュールや SWC（AUTOSAR ソフトウェアコンポーネント）[1]、ファンクションを割り当て、さらにモジュール／ SWC ／ファンクション間の接続情報を定義します。実行シーケンスは、INTECRIO V5.0 においては全体のコンフィギュレーションに基づいて自動的に決定されます。

「環境システム」（Environment System）は、仮想プロトタイピングで使用するプラントモデル（制御対象モデル）です。プラントモデルには、ドライバ、車両、環境の各モデルが含まれます。環境システムの構築は、ソフトウェアシステムと同じ方法で行います。

ソフトウェアシステムの各コンポーネントは、`Software` フォルダの各サブフォルダに格納されます。 同様に、 環境システムのコンポーネントは `Environment` フォルダの各サブフォルダに格納されます。

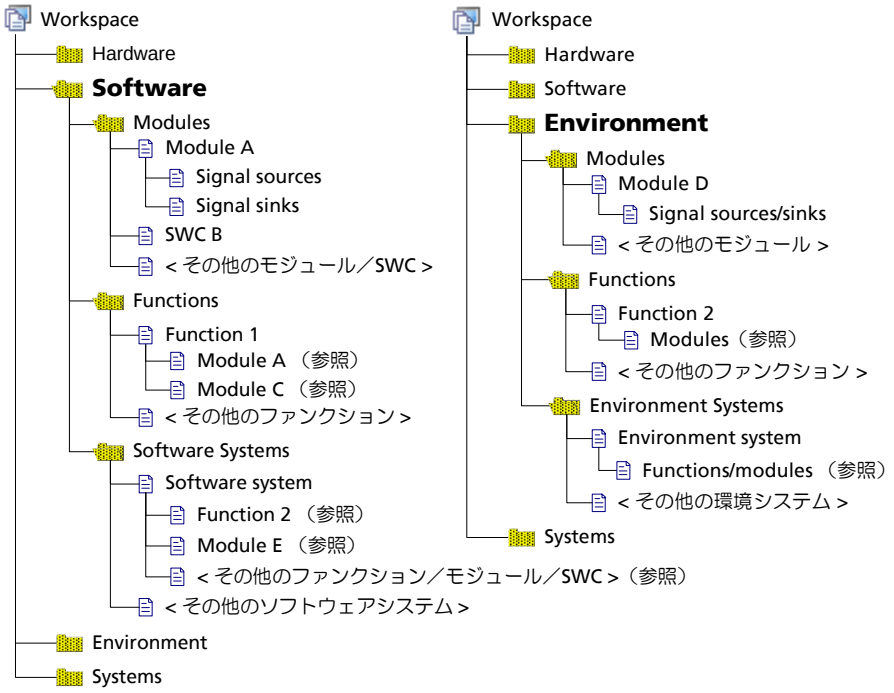


図 3-21 フォルダ構造：ソフトウェアシステムと環境システム

1. 詳細は、3.5.1.1 項「モジュールと AUTOSAR ソフトウェアコンポーネント（SWC）」を参照してください。

システムプロジェクト

「システムプロジェクト」は ECU ソフトウェア全体を定義するもので、ハードウェアシステム、ソフトウェアシステム、環境システム（仮想プロトタイプのみ）を 1 つずつ割り当てて構築されます。また、ハードウェアシグナルとソフトウェアシグナルとのマッピングや、オペレーティングシステムのコンフィギュレーション（「OS コンフィギュレーション」）もこのシステムプロジェクト内に定義されます。システムプロジェクトをビルドすることによってプロトタイプ（プロトタイピング実験用の実行ファイル）が生成されます。

システムプロジェクトは、`System` フォルダ内に格納されます。

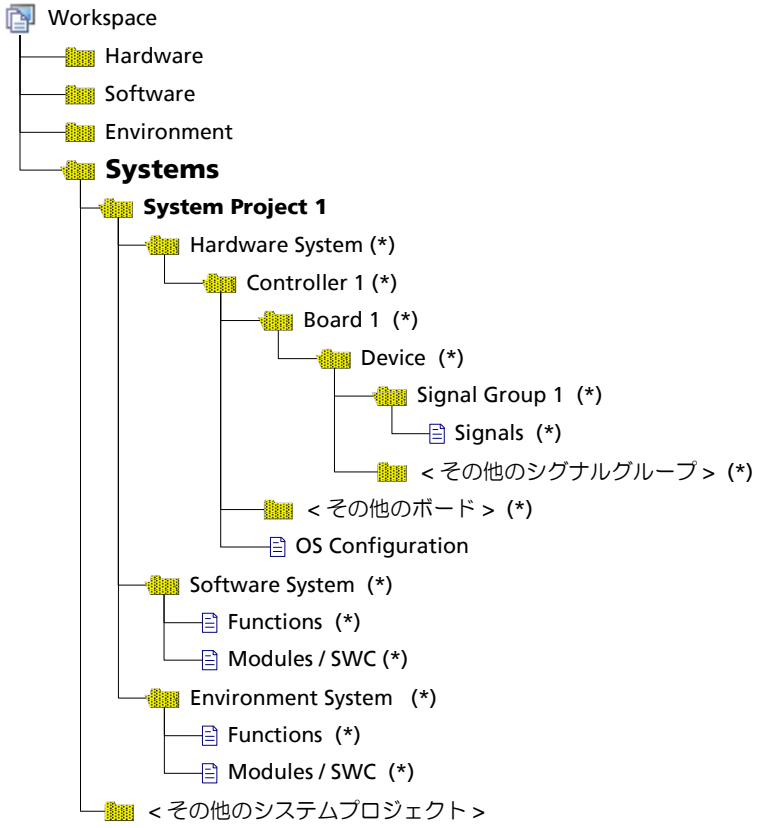


図 3-22 フォルダ構造：システムプロジェクト（* の付いたアイテムは参照されるアイテムです）

ソフトウェアシステム、ハードウェアシステム、環境システム（仮想プロトタイピング用）、システムプロジェクトについては以降の項で詳しく説明します。

3.5.1 ソフトウェアシステム

本項では、ソフトウェアシステムについて詳しく説明します。図 3-23 は、ソフトウェアシステムの構造を示しています。アイテム間の実線は内包関係を表し、背景に破線で描かれているアイテムは、必要に応じて追加できるオプションアイテムを表します（これらのオプションアイテムは、それぞれの前面に実線で描かれているアイテムと同じタイプのアイテムを内包します）。

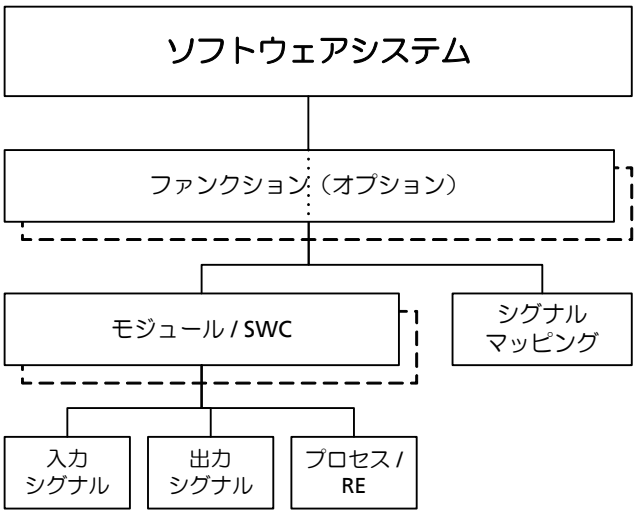


図 3-23 ソフトウェアシステムの構造

3.5.1.1 モジュールと AUTOSAR ソフトウェアコンポーネント（SWC）

注記

INTECRIO V5.0.2 における AUTOSAR のサポートは限定的なものになります。従来の AUTOSAR モジュールを含む既存のワークスペースはサポートされますが、AUTOSAR ソフトウェアコンポーネントを新たにインポートすることはできません。

INTECRIO で扱われる「モジュール」および「AUTOSAR ソフトウェアコンポーネント（SWC）」は、BMT で作成されたソフトウェアモジュールについてのインターフェースディスクリプションをインポートしたものです。

モジュールとしては、ユーザー定義された ASCET モジュールや Simulink モデル、AUTOSAR ソフトウェアコンポーネントのほか、C コードのテストファンクションや複雑なスティミュラスジェネレータ、さらにはプラントモデル（「ソフトウェアインザループ」アプリケーション用）も使用できます。モジュールを作成するには、「SCOOP-IX」（6 項「SCOOP と SCOOP-IX」参照）というインターフェースディスクリプションファイル、または XML ファイル（AUTOSAR ソフトウェアコンポーネントの場合）をインポートします。

モジュールと SWC の構成の概略は 20 ページの図 3-10 に示されています。INTECRIO が個々のモジュールについて認識するのは、ユーザー定義されたモジュールまたは SWC のインターフェースディスクリプションファイルに記述されたシグナルソースとシグナルシンク（入力と出力）、および起動インターフェース（プロセス、または SWC の場合はランナブルエンティティ）だけです（図 3-14 参照）。各種 BMT で作成されたソフトウェアモジュールや SWC の機能自体は、プロトタイプのランタイムにのみ意味を持ちます。

また INTECRIO では、ハードウェア I/O ポートもシグナルソースとシグナルシンクを備えた「モジュール」として認識されます。

モジュール間の情報交換を可能にするためには、「シグナルマッピング」を定義し、各モジュールの入力と出力を明示的に接続する必要があります。暗黙的な接続（同名シグナルの自動接続など）は行われません。

図 3-24 に簡単な接続例を示します。1 つのソースを 1 つのシンク（A）または複数のシンク（B）と接続します。ソースとシンクにおける入出力のタイミングは同じです。

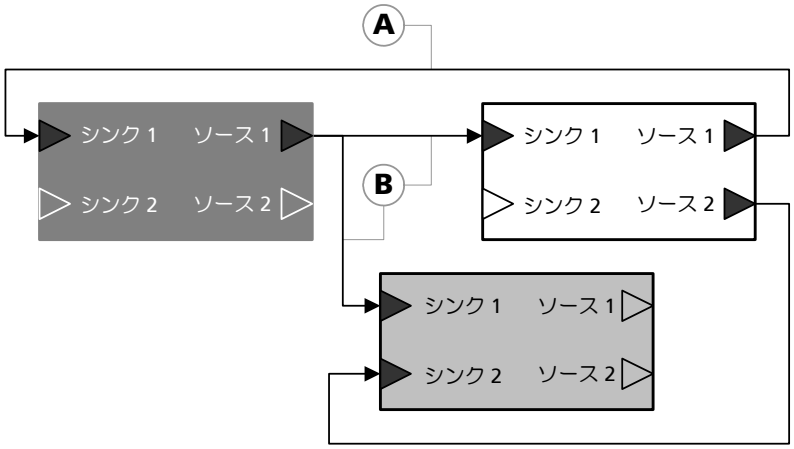


図 3-24 接続：1 つのソースから 1 つまたは複数のシンクへ同時に入出力

さらに以下の例のようなさまざまな接続が可能です。

- 1 つのソースを複数のシンクに接続し、ソースとシンクの入出力タイミングを変える場合

図 3-25 接続：1 つのソースから複数のシンクへタイミングをずらして入出力

この場合、ソースのデータがバッファリングされた後、シンクに送られます。

- 2 つのソースを 1 つのシンクに接続し、ランタイムにおいて切り替える（ラピッドプロトタイピングの場合のみ）

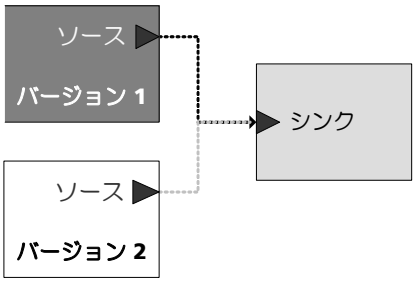


図 3-26 接続：複数のソースから 1 つのシンクへの接続

図 3-26 の接続は、プロトタイピング実験の実行中にクロスバーマネージャを用いて 2 つのソースを切り替える方法（3.5.5 項参照）を表しています。また、ソフトウェア開発フェーズでこのような接続を恒久的にコーディングする方法もあります。

3.5.1.2 ファンクション

ソフトウェアシステムを定義する際、ソフトウェアシステムに個々のモジュールと SWC を直接割り当てることもできますが、関連する複数のモジュールを「ファンクション」として定義しておき、ファンクション単位でモジュールをソフトウェアシステムに割り当てることもできます。これにより、ソフトウェアシステムの内容がより把握しやすくなります。

ファンクションは、ASCET の「階層ブロック」と同様の、固有の機能を持たない「分類オブジェクト」（"classification object"）です。

ファンクションは以下のコンポーネントで構成されます。

- 1 つまたは複数のモジュール／ SWC
- ファンクション内の各モジュール／ SWC の入出力間の接続
- ファンクションインターフェース（入力、出力、起動インターフェース）

ファンクションの入力と出力には、データや実装情報を定義することはできません。ファンクションの入力は、ファンクション内のモジュール／ SWC の複数のシンクに接続できますが、出力への接続は、ファンクション内の 1 つのソースからしか行えません。SWC については、そのインターフェースタイプに応じて、1 つのポートから 1 つの出力にのみ接続できるか、または複数の出力に接続できるかが決まります（46 ページ「ポートとインターフェース」参照）。

また必要に応じて、ファンクションの入力と出力を自動的に作成することもできます。この場合、入力については、まだ接続されていないモジュール／ SWC のシンク／レシーバのみが考慮されます。また出力については、ファンクション内の全モジュール／ SWC のすべてのソース／センダ、または未接続ソース／センダのみが考慮されます。

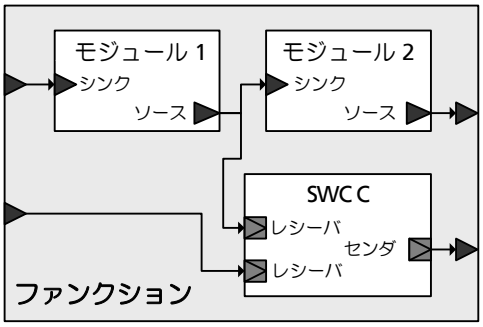


図 3-27 ファンクションの構成例

INTECRIO ではモジュールをインスタンス化してファンクションに組み込むことはできません。つまり、1 つのファンクションに同じモジュールを 2 回挿入することはできません。

3.5.1.3 ソフトウェアシステム

「ソフトウェアシステム」は ECU ソフトウェアのアプリケーション部分全体に対応し、任意の数のファンクションと個々のモジュール／ SWC とを組み合わせて作成します。

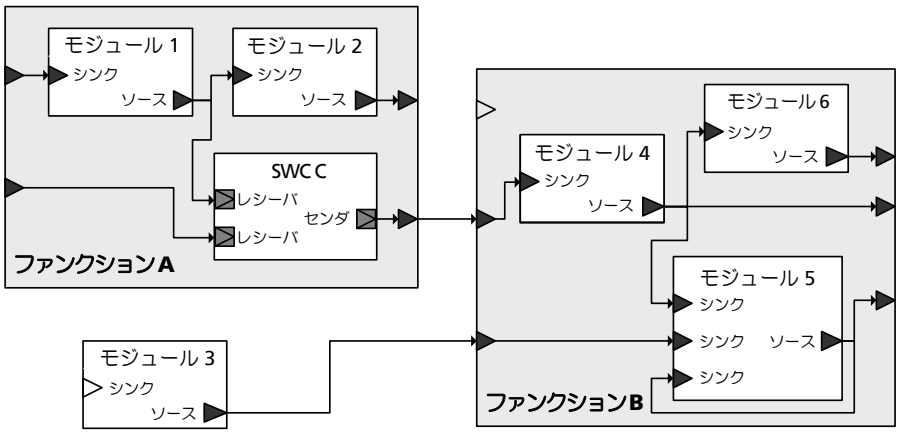


図 3-28 ソフトウェアシステムの例

モジュール／ SWC を「インスタンス化」してソフトウェアシステムに組み込むことはできません。つまり、各モジュール／ SWC は、ファンクションに内包されているものも含め、1 つのソフトウェアシステム内には 1 回しか挿入できません。なお、マルチインスタンスのチェックはコード生成時にしか行われません（3.5.4 項参照）。

注記

すでにファンクションの一部としてソフトウェアシステムに含まれているモジュール／ SWC を、さらに単独でソフトウェアシステムに挿入することは避けてください。そのような操作を行うと、コード生成時に以下のエラーメッセージが発行されます。

```
Build preparation error: Multiple instances of same
module in software system
```

3.5.2 環境システム

環境システムは、仮想プロトタイピング（3.3 項参照）に用いられるプラントモデルをモデリングするためのものです。環境システムは、ソフトウェアシステムと同様にモジュール／ SWC とファンクションで構成されます。

1 つのモジュール／ SWC またはファンクションは、ワークスペース内においてソフトウェアシステムまたは環境システムのいずれか一方でのみ使用できます。つまり、たとえば `Software` フォルダ内にインポートして作成されたモジュール （`Software/Modules`） やファンクション （`Software/Functions`）を環境システムに組み込むことはできません。

3.5.3 ハードウェアシステム

図 3-29 はハードウェアシステムの構成を示しています（アイテム間の実線は内包関係を示します）。ハードウェアシステムには、各ハードウェアと、インターフェース（ETK や CAN など）についてのすべての定義が含まれます。

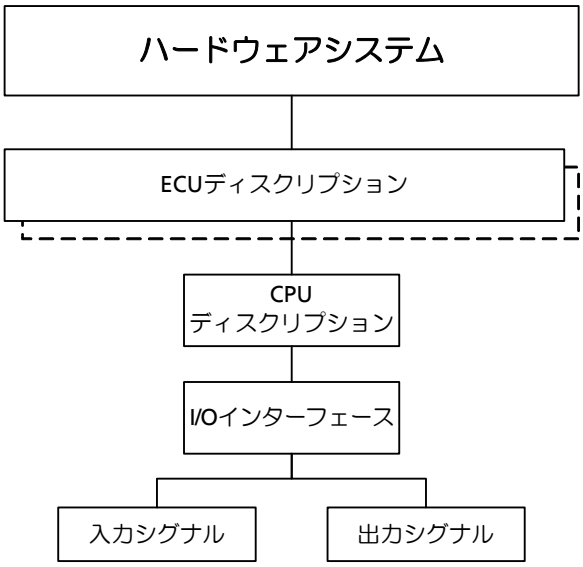


図 3-29 ハードウェアシステムの構造

「ECU ディスクリプション」は、1 台の ECU に相当する 1 つのハードウェアシステムについて定義するものです。実験ターゲットは、ボード構成やインターフェースが異なっていても「ECU」として認識されます。

ECU ディスクリプションには、ECU 内のプロセッサや I/O ハードウェア、およびそれらを接続するインターフェースについての記述が含まれます。

「CPU ディスクリプション」は個々のプロセッサの詳細情報を記述したものです。これには以下の情報が含まれます。

- プロセッサの特性（タイプ、ブランドなど）
- プロセッサ速度
- メモリレイアウト
- その他のプロセッサ固有の設定情報

「I/O インターフェース」は、入出力信号とプロセッサのインターフェースについて記述されたものです。

「入力シグナル」および「出力シグナル」は、各 I/O ハードウェアの入出力信号に対応するシグナルです。ハードウェア（センサやアクチュエータ用）の入出力信号とソフトウェアモデルのシグナルを接続するために使用されます。

3.5.4 システムプロジェクト

「システムプロジェクト」は、1 つのハードウェアシステムと 1 つのソフトウェアシステム、および 1 つの環境システム（仮想プロトタイピングの場合のみ）が統合されたものです。各モジュールは、1 つのプロジェクト中でそれぞれ 1 回ずつしか使用できないので、複数回使用されているモジュールがあると、ビルド時にエラーメッセージが発行されます。

システムプロジェクト内では、ハードウェアの入出力信号に対応するハードウェアシグナルが、ソフトウェア内で定義されたシグナルにマッピングされ、さらに各プロセスの処理シーケンスが「OS コンフィギュレーション」として定義されます。

図 3-30 はシステムプロジェクトの構造を示したものです。背景に破線で描かれているアイテムは、必要に応じて追加できるオプションアイテムを表します（これらのオプションアイテムは、それぞれの前面に実線で描かれているアイテムと同じタイプのアイテムを内包します）。

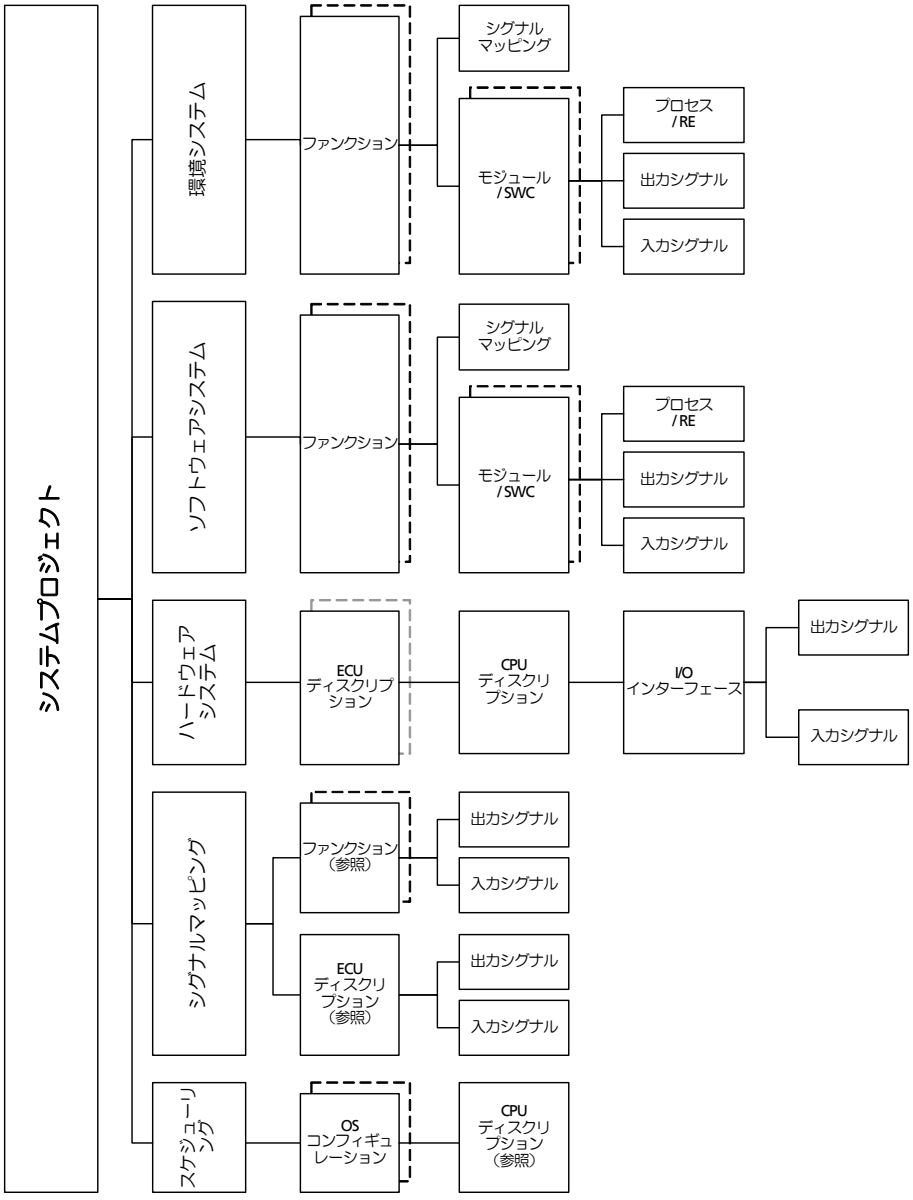


図 3-30 システムプロジェクトの構造

上記のアイテムのうち、ソフトウェアシステム、環境システム、ハードウェアシステムについては 3.5.1 項、3.5.2 項、3.5.3 項ですでに説明されています。

「シグナルマッピング」は、ソフトウェアまたは環境のファンクション／モジュール／ SWC と ECU ディスクリプションの入出力を「参照」し、それらをマッピングするものです。ソフトウェアシステムやハードウェアシステム自体の内容が変更されると、その変更内容は直ちにシグナルマッピングに反映されます。

「スケジューリング」、つまりタスクとプロセス／ RE（ランナブルエンティティ）の処理シーケンスは、オペレーティングシステムの設定として「OS コンフィギュレーション」内に定義されます。OS コンフィギュレーション内でタスク生成と設定を行い、各タスクにシステムのプロセス（起動インターフェース）を順に割り当てます。これらの操作は、自動で行うこともできます。

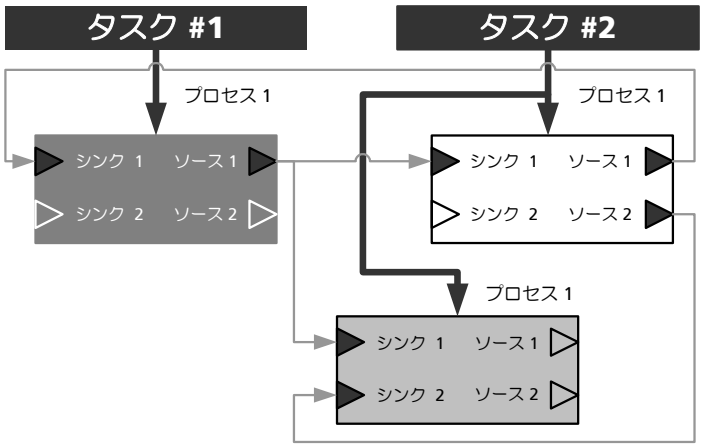


図 3-31 プロセスをタスクに割り当てる

3.5.5 クロスバー（Crossbar）

モジュール、ファンクション、ハードウェア間の接続は、「クロスバー」（Crossbar）により管理されます。ただしシステムプロジェクト内に 1 つ以上の SWC が含まれる場合は、クロスバーの代わりに「AUTOSAR ランタイム環境（RTE：**R**un**t**ime **E**nvironment）」（45 ページの 4.2 項参照）が使用されます。

クロスバーは、実験ターゲット（ES830 など）上で実行されるプロトタイプソフトウェアを構成するソフトウェアコンポーネントの 1 つで、ランタイムにおいてモジュール、ファンクション、ハードウェアの間の接続（具体的には、シグナルソースとシグナルシンクの接続）を管理し、調整する役割を持ちます。この際、1 つのシグナルソースを複数のシグナルシンクに接続することはできますが、複数のシグナルソースを 1 つのシグナルシンクに接続することはできません。

クロスバーは、プロジェクトコンフィギュレータからの情報に基づいて「静的コネクション」および「動的コネクション」という 2 種類の接続を扱うことができます。静的コネクションの場合、定義されたルーティングをランタイムに変更することはできないので、変更の必要が生じた場合は、実験を中止し、接続の定義を変更してプロジェクトをビルドしなおす必要があります。これに対して、動的コネクションの場合は、ランタイムにルーティングを切り替えることができます。

プロトタイプのランタイムにおいて、クロスバーは各接続についての情報（ソース、シンク、プロセス、変数タイプなど）を受け取り、その情報に従って受け渡されるデータに値を代入します。この代入は、必ず、その値を受け取るモジュー

ルが呼び出される前に行われます。このようにしてシグナルソースとシグナルシンクの接続が確立されます（図 3-32 の灰色の円で示された箇所が接続ポイントを表します）。

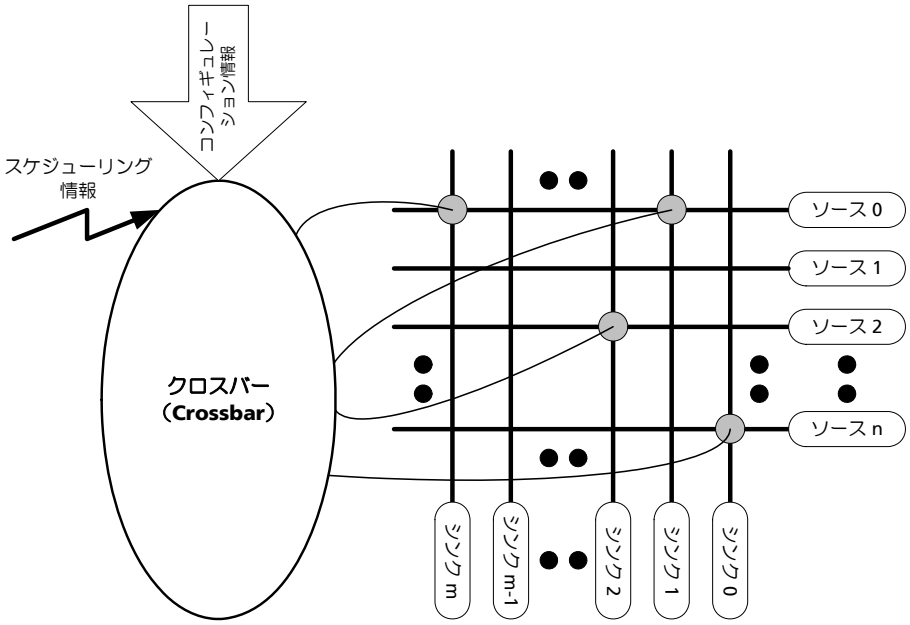


図 3-32 クロスバーの機能概要

特許により保護されているクロスバーは、入念に設計されたコピーロジックにより、モジュールのリアルタイム性を保証します。この際、必要に応じて量子化やデータ型が自動的に変換されます。

この変換処理を、以下に、`sink 1 = source 0` というシンプルな代入式を例にして説明します。

代入式の左辺と右辺の量子化が異なる場合、`source 0` の値の量子化が変更されてから `sink 1` に代入されます。 また左辺と右辺のデータ型が異なる場合は、 `source 0` が `sink 1` のデータ型に変換されます。

3.6 プロトタイプの実験

「プロトタイプ」は、プロジェクトインテグレータ（5.9 項「プロジェクトインテグレータ」参照）によってシステムプロジェクトから生成される実行ファイルです。プロトタイプには ECU ソフトウェアの制御アルゴリズムが実行可能なコードとして書き込まれていますが、その構成と目的は、最終的な製品用のものとは異なります。

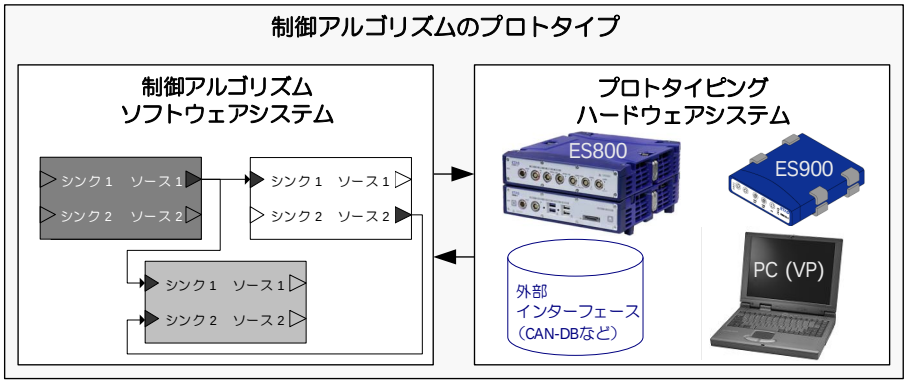


図 3-33 ラピッドプロトタイピングと仮想プロトタイピングの実験用プロトタイプ

プロトタイプは、ETAS の実験ツール「ETAS Experiment Environment」を介してラピッドプロトタイピングハードウェア上で実行することにより、リアルタイム条件下での実験を行うことができ、必要に応じて自動実行モードに設定することも可能です。この実験は、以下の目標を達成するためのものです。

- 制御アルゴリズム全体またはその一部分の評価
- 実装されたソフトウェアの検証（動作テスト）
- モデル全体の実装条件の検証

実験では、測定／適合ウィンドウを用いて値の測定や適合を行うことができます。これらのウィンドウの設定（変数の追加・削除、表示設定の変更など）は実験中でも行えます。また PC ターゲットによる仮想プロトタイピング実験では、シミュレーション速度を実験中に調整することもできます。

「動的コネクション」として作成された接続は、実験の実行中にルーティングを動的に切り替えることができます。この機能は、たとえば固定小数点コードで実装されたモデルの挙動を、他のモデル（物理モデルなど）の挙動と比較するような場合に、非常に便利です。

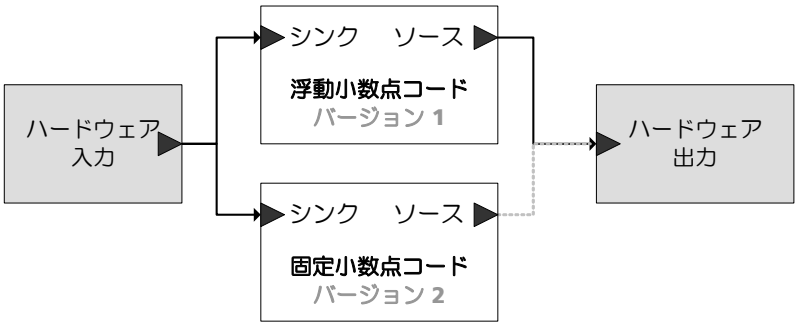


図 3-34 実行されるモデルを動的に切り替える

その他、実験の実行中に以下のような操作を行えます。

- 単純な I/O シグナルの変換式の変更

- 各シグナルグループのオン／オフ切り替え
- 各イベントのオン／オフ切り替え

実験中に得られるデータはデータロガーによって記録することができ、記録されたデータはファイルに保存されます。

4 INTECRIO と AUTOSAR

本章では INTECRIO の AUTOSAR 対応について説明します。4.1 項では AUTOSAR の趣旨を概説し、続いて 4.2 項では AUTOSAR のランタイム環境（RTE）について説明し、4.3 項で INTECRIO がサポートする AUTOSAR エレメントを紹介します。

注記

INTECRIO V5.0.2 における AUTOSAR のサポートは限定的なものになります。従来の AUTOSAR モジュールを含む既存のワークスペースはサポートされますが、AUTOSAR ソフトウェアコンポーネントを新たにインポートすることはできません。

本章では AUTOSAR 自体についての詳しい説明は省略します。詳細な情報が必要な場合は AUTOSAR のウェブサイト（https://www.autosar.org/）をご覧ください。

4.1 概要

一般的に、自動車組み込みソフトウェアの開発工程において、複数のベンダーから提供されるソフトウェアコンポーネントを複数のネットワークや ECU（電子制御ユニット）、各種ソフトウェアアーキテクチャで構成される車両プロジェクトに統合するには、非常な手間がかかります。また、ソフトウェアの「再利用性」に制限があれば、高機能を持った検証済みの高品質なソフトウェア を提供するためには多大な工数が必要となります。

このような状況の中で、車両メーカーとベンダーとのパートナーシップにより組織された「AUTOSAR」では、主に基本的なシステムファンクションとファンクションインターフェースを標準化することにより、車両エレクトロニクス用ソフトウェアの共同開発の簡素化、費用低減、工期短縮、品質向上などを図り、さらには安全関連システムの設計に必要なメカニズムの提供も目指しています。

これらの目標を達成するため、AUTOSAR では自動車組み込みソフトウェア用のアーキテクチャを定義しています。このアーキテクチャには、各アプリケーションの機能を実装する「ソフトウェアコンポーネント」（**SWC**：AUTOSAR **S**oftware **C**omponent）が含まれていますが、この SWC は ECU に依存しないものであるため、再利用、交換、スケールの変更、統合が容易に行えます。

このように抽象化された SWC 環境は、「仮想ファンクションバス」（**VFB**：**V**irtual **F**unction **B**us）と呼ばれています。実際の AUTOSAR 対応の ECU 内においてこの VFB は、各 ECU に依存するプラットフォームソフトウェアにマッピングされます。AUTOSAR「プラットフォームソフトウェア」の機能は、「ランタイム環境」（**RTE**：**R**un **T**ime **E**nvironment）と「基本ソフトウェア」（**BSW**：**B**asic **S**oftware）とに分けられます。BSW に含まれるものは、通信や I/O の機能のほか、各種ソフトウェアコンポーネントに必要な諸機能（診断やエラー報告、不揮発メモリ管理など）です。

4.1.1 RTA-RTE と RTA-OS

「ランタイム環境」は、ソフトウェアコンポーネント、BSW モジュール、オペレーティングシステム（OS）の間のインターフェースとなるものです。SWC の相互接続においては、RTE は通信の「スイッチボード」のような役割を果たします。この役割は、コンポーネントが単体の ECU に常駐する場合でも、車載バスによりネットワーク化された複数の ECU に常駐する場合でも同じです。

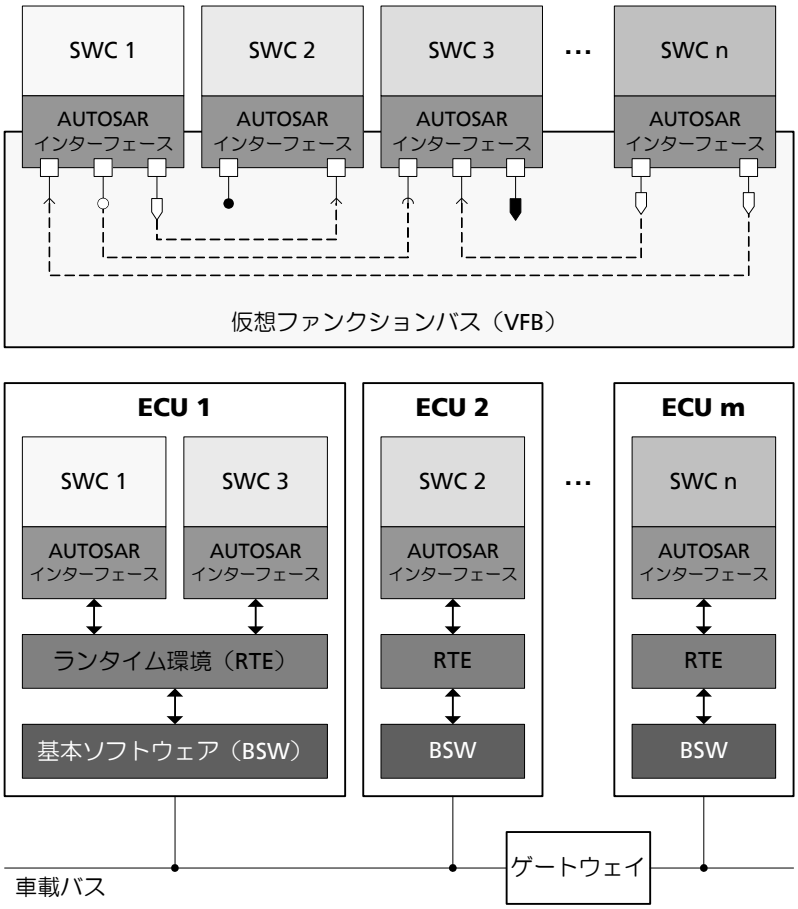


図 4-1 AUTOSAR ソフトウェアコンポーネント（SWC）の通信は、「ランタイム環境（RTE）および基本ソフトウェア（BSW）」によって実現される「仮想ファンクションバス（VFB）」経由で行われます。

AUTOSAR では、OS は RTE を通じて SWC のランナブルエンティティを呼び出します。アプリケーションソフトウェアの実行制御に関しては、基本ソフトウェアの中でも RTE と OS がキーモジュールとなります。

ETAS は、「ERCOS^{EK}」に始まり、そして「RTA-OSEK」と、長年にわたり車両用オペレーティングシステムを自動車業界に供給してきました。AUTOSAR ランタイム環境「RTA-RTE」と AUTOSAR オペレーティングシステム「RTA-OS」は、AUTOSAR の主要ソフトウェアモジュールをサポートし、RTA の機能範囲を拡張します。

AUTOSAR インターフェースに準拠する RTA-RTE および RTA-OSEK は、サードパーティベンダーの基本ソフトウェアモジュールをシームレスに統合することが可能です。

4.1.2 AUTOSAR ソフトウェアコンポーネントの作成（INTECRIO の機能範囲外）

INTECRIO には AUTOSAR ソフトウェアコンポーネントの作成を行う機能は含まれていませんが、ASCET を使用すれば AUTOSAR に準拠する車両機能の挙動を定義して実装することができます。また必要に応じて、システムアーキテクチャや AUTOSAR インターフェースのディスクリプションのたたき台を作成するための AUTOSAR オーサリングツールもご利用いただけます。

ASCET モデルは、容易に AUTOSAR に適応させることができます。これは、AUTOSAR コンセプトの多くの部分を ASCET のインターフェース記述と同じような形式で定義することができるためです。実際に行う作業としては、各アプリケーションのインターフェースを AUTOSAR に適合するように修正するだけです。従来のモデルを実際に適合させた事例からわかるように、現行バージョンの ASCET バージョンでこの作業を行っても、それほど多くの所要時間は要しません。

ASCET の各バージョンは、それぞれ所定のバージョンの AUTOSAR を対象として、AUTOSAR SWC ディスクリプションの作成と SWC 量産コードの生成を行います。AUTOSAR の対応バージョンについての詳細は ASCET のユーザードキュメントを参照してください。

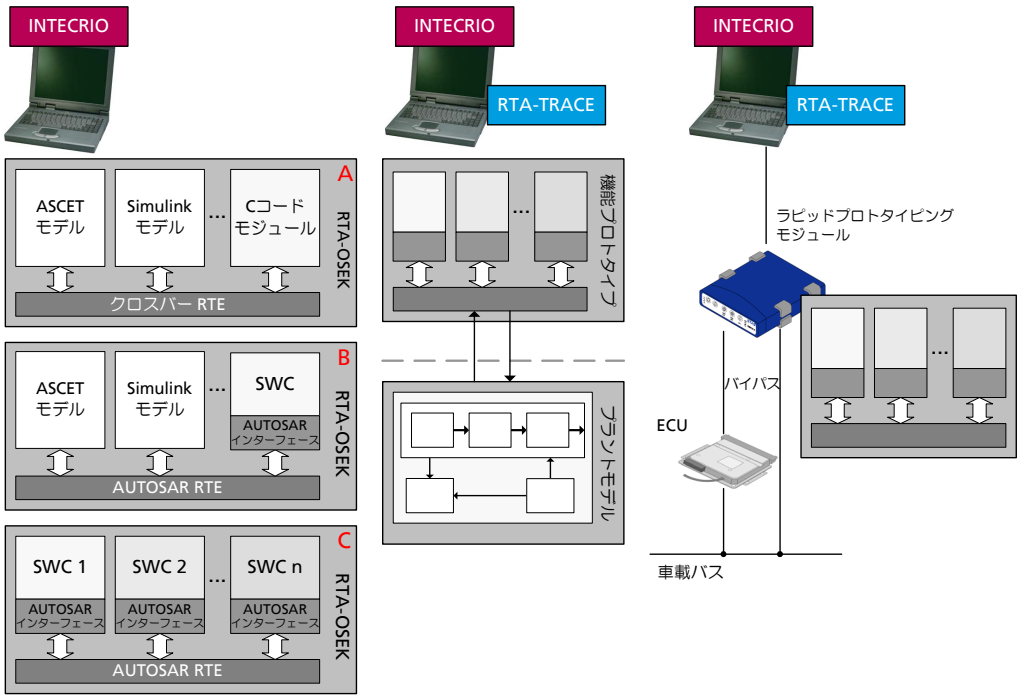
4.1.3 ソフトウェアコンポーネントの検証

AUTOSAR の「仮想ファンクションバス」という概念により、「仮想統合」への道が開かれました。仮想ファンクションバスにより ECU 境界が曖昧になるため、設計段階において各種ファンクション用のソフトウェアコンポーネントの統合を行う際、各 ECU への最終的な割り当てを行う必要はなくなります。そのため、RTE により統合されるソフトウェアコンポーネント同士の相互的な処理状態を、AUTOSAR OS が稼働する PC 上で容易に検証できます。

INTECRIO は車載エレクトロニクスシステムのプロトタイピングと検証を行うための優れた環境を提供します。INTECRIO バージョン V5.0 を使用することにより、レガシー AUTOSAR SWC と機能モジュールを統合することができます（図 4-2）。これにより、ECU ソフトウェアを AUTOSAR アーキテクチャに移行する過程において、既存のモデルや C コードを再利用することが可能となります。

レガシー AUTOSAR SWC は、ファンクション（図 4-2 の **B**）に組み込んだり、純粋な AUTOSAR システム（図 4-2 の **C**）としてテストしたりすることができます。

プラントモデルを使用すれば、PC 上で Model-in-the-Loop 実験を行えます。



ステップ 1:
ソフトウェアコンポーネントの統合

ステップ 2:
PC上での仮想プロトタイピング

ステップ 3:
ハードウェア環境でのラピッドプロトタイピング

図 4-2 仮想プロトタイピング用（図中央）またはラピッドプロトタイピング用（図右）のソフトウェアモジュールに、AUTOSAR RTE を統合（図左）。

INTECRIO V5.0 は表 4-1 に示される AUTOSAR バージョンについて、レガシー AUTOSAR SWC ディスクリプションの使用と RTE コンフィギュレーションの生成をサポートします。

AUTOSAR バージョン	注記
R3.0.0	ASCET はサポートしていません
R3.0.1	ASCET はサポートしていません
R3.0.2	
R3.1.0	
R3.1.2	
R3.1.4	
R4.0.2	INTECRIO V5.0 は NVData インターフェースと ModeSwitch インターフェースをサポートしていません。
R4.0.3	

表 4-1 サポートされている AUTOSAR バージョン

INTECRIO V5.0 は ASCET-SE で生成された AUTOSAR SWC ディスクリプション用の補間ルーチンも提供します。

INTECRIO V5.0 では、AUTOSAR R4.0 用の RTA-RTE[1]（表 4-1 参照）と RTA-OSEK V5.0 との併用により、量産用のものに非常に近い状態の AUTOSAR プロトタイプを PC 用および ETAS ラピッドプロトタイピングシステム用に作成することができます。

1. RTA-RTE 3.1: AUTOSAR R3.1.4、R3.1.2、R3.1.0、R3.0.0、R3.0.1、R3.0.2
RTA-RTE 5.4: AUTOSAR R4.0.*

INTECRIO においてはソフトウェアコンポーネント間通信とプロトタイピングハードウェアコンフィギュレーションとが厳密に区別されているため、INTECRIO V5.0 では、検証済みの RTE コンフィギュレーションを XML ファイル形式でエクスポートできます。この情報を用いることにより、RTA-RTE などの AUTOSAR RTE ジェネレータで AUTOSAR ECU の RTE を作成することができます。

4.2 ランタイム環境とは

VFB（仮想ファンクションバス）は、ソフトウェアコンポーネントの再利用化のための抽象概念です。「RTE」（ランタイム環境）は、この VFB の機能を実行するために必要なメカニズムを提供します。最もシンプルな RTE は、VFB を実装するだけでできあがりますが、実際には、各種インターフェースやインフラストラクチャを用意して、ソフトウェアコンポーネントを以下のように扱えるようにする必要があります。

A ECU（VFB モデル）に依存せずに SWC を RTE に実装できること

B. アプリケーションソフトウェア自体を修正することなく、ECU および大規模な車両ネットワークに SWC を統合できること（Systems Integration モデル）。

具体的には、RTE には以下のような機能が必要となります。

- ソフトウェアコンポーネント用通信インフラストラクチャ

  ここでいう「通信」には、同一 ECU 上のソフトウェアコンポーネント間の通信（「ECU 内通信」）と、異なる ECU 上のソフトウェアコンポーネント間の通信（「ECU 間通信」）との両方が含まれます。

- ソフトウェアコンポーネントのリアルタイムスケジューリング

  AUTOSAR におけるリアルタイムスケジューリングとは、SWC のランナブルエンティティを、設計時に定義された時間的条件に基き、オペレーティングシステムが提供するタスクにマッピングすることを指します。

BSW（基本ソフトウェア）は、RTE により実装される「抽象概念」の下層に位置するため、アプリケーションソフトウェアコンポーネントから直接アクセスすることはできません。つまり、SWC がオペレーティングシステムや通信サービスなどに直接アクセスすることはできません。そのため RTE は、それらのサービスをカバーする「抽象概念」を提供する必要があります。この抽象概念は、ソフトウェアコンポーネントがどこに配置されていても常に不変でなければなりません。この条件に基づき、ソフトウェアコンポーネント間のすべての通信は、標準化された RTE インターフェースの呼び出しにより行われることになります。

さらに RTE は、1 つまたは複数の ECU に配置された SWC で構成される既存の「アーキテクチャ」を実際に利用する際にも使用されます。RTE を効果的に実装するために、アーキテクチャに必要な RTE のコードはビルド時において各 ECU ごとに決定されます。標準化された RTE インターフェースの実装は、RTE 生成ツールにより自動的に行われますが、その際、指定されたコンポーネントインタラクションとコンポーネントアロケーションを実現するための適切なインターフェースが生成されます。

同じ ECU 上に配置された 2 つのソフトウェアコンポーネント間の通信には「ECU 内通信」を使用できますが、どちらか一方を別の ECU に移した場合、両者間の通信には「車両ネットワーク」が必要となります。

このようにして生成される RTE により、各 ECU 用の基本ソフトウェアの差異は、以下のようにアプリケーションソフトウェアコンポーネントから隠蔽されることになります。

- 一貫したインターフェースをソフトウェアコンポーネントに提供することにより、ソフトウェアコンポーネントの再利用が可能となります。つまり、一度設計して作成されたソフトウェアコンポーネントを複数の ECU に再利用できます。
- このインターフェースは、VFB 内に「抽象概念」として実装された AUTOSAR 基本ソフトウェアにバインドされます。

4.3 INTECRIO がサポートする AUTOSAR エレメント

INTECRIO は以下の AUTOSAR エレメントをサポートしています。

4.3.1 AUTOSAR ソフトウェアコンポーネント

「AUTOSAR ソフトウェアコンポーネント」（SWC: AUTOSAR **S**oft**w**are **C**omponent）はアプリケーションレベルの汎用コンポーネントで、CPU 内、および車両ネットワーク内の配置に依存しません。システム設計者が定義した制約条件に従い、システム設定時において任意の ECU 上に配置することができます。

ただし SWC は AUTOSAR システムを分散化する際の最小単位であるため、1 つのコンポーネントは必ずその全体を 1 つの ECU 上に配置する必要があります。

SWC を作成する際は、そのコンポーネントタイプ（「SWC タイプ」）を定義します。この「SWC タイプ」は、SWC の固定的特性（ポート名や、ポートがインターフェースによりどのように使用されるか、などの情報）を定義するものです。SWC タイプには、システム内で一意の名前を付ける必要があります。これらの点をふまえ、各 SWC は以下のもので構成されます。

- 形式に従って定義された SWC ディスクリプション — コンポーネントのインフラストラクチャの設定方法を定義するもの
- SWC の実装コード — C コードによる SWC の機能記述

SWC は、システム設定時において「インスタンス化」することにより、使用可能となります。ここでの「タイプ」と「インスタンス」は、従来のプログラミング言語における「型」と「変数」の関係と似ています。つまり、従来のアプリケーション内で一意の型名（つまり SWC タイプ）を定義し、その型を用いて、一意の名前を持つ変数（1 つの SWC インスタンス）を宣言するのと同じ要領です。

4.3.2 ポートとインターフェース

VFB モデルにおいて SWC は、インターフェースによってアクセスされる「ポート」を通じて他のコンポーネントとの通信を行います。「*インターフェース*」は、通信によって伝達される情報や通信のセマンティックスを制御します。SWC はポートを通じてインターフェースにアクセスし、この「ポート」と「ポートインターフェース」を組み合わせたものを「AUTOSAR インターフェース」と呼びます。

ポートには以下の 2 種類があります。

- 提供ポート（P ポート：Provided port）— SWC はこのタイプのポートを使用してデータやサービスを他の SWC に提供します。P ポートには「センダポート」と「サーバーポート」とがあります。

- 要求ポート（R ポート：Required port）— SWC はこのタイプのポートを使用して他の SWC にデータやサービスを要求します。R ポートには「レシーバポート」と「クライアントポート」とがあります。

> **注記**
>
> 本書では、非 AUTOSAR ポートとの混同を避けるため、AUTOSAR ポートを「R ポート」および「P ポート」と呼びます。

INTECRIO V5.0 は、既存のレガシー AUTOSAR モジュール[1] について、以下の 3 種類のインターフェースをサポートします。

- センダ／レシーバ（シグナル渡し）
- クライアント／サーバー（ファンクション呼び出し）
- 適合パラメータインターフェース

SWC に他のインターフェース（NVData インターフェースや ModeSwitch インターフェースなど）が含まれている場合、これらのインターフェースはインポート時にスキップされ、その旨が警告メッセージとして通知されます。

SWC の各 P ポートおよび R ポートには、その SWC が提供または要求するインターフェースタイプを定義します。

1 つのシステムプロジェクトが複数の SWC インスタンスにより構築されている場合、それらのインスタンスの R ポートと P ポートを接続します。この際、センダをレシーバに接続し、クライアントをサーバーに接続します。

4.3.2.1 センダ／レシーバ通信

センダ／レシーバ通信においては、最小単位のデータエレメントで構成されるシグナルを 1 つの SWC が送信し、1 つまたは複数の SWC がそのシグナルを受信します。

1 つの SWC タイプに複数の「センダ／レシーバポート」を持たせることができます。

各センダ／レシーバポートには、個別に送受信できる複数のデータエレメントを定義することができます。インターフェース内のデータエレメントとしては、単純データ型（integer や float など）や複合データ型（array や matrix など）を使用できます。

センダ／レシーバ通信は単方向通信なので、レシーバからの応答が必要な場合は、その応答を別のセンダ／レシーバ通信としてモデリングする必要があります。

センダ／レシーバインターフェースを要求する SWC の R ポートはインターフェースのデータエレメントを読み取ることができます。一方、センダ／レシーバインターフェースを提供する P ポートはデータエレメントを書き込むことができます。

上記のように、INTECRIO V5.0 のセンダ／レシーバ通信においては "1:n"（1 つのセンダ、複数のレシーバ）の組み合わせのみ可能です。"n:1"（複数のセンダ、1 つのレシーバ）という組み合わせはサポートされていません。

1.「レガシー AUTOSAR モジュール」とは、V5.0.0 以前の INTECRIO にインポートされた AUTOSAR モジュールを指します。

4.3.2.2 クライアント／サーバー通信

クライアント／サーバー通信においては、ある SWC の中に定義されているサーバーファンクションを別の SWC が呼び出します。この際、前者の SWC は応答を返す場合も返さない場合もあります。

1 つの SWC タイプに複数のクライアント／サーバーポートを持たせることができます。

各クライアント／サーバーポートには、個別に呼び出せる複数の「オペレーション」を定義できます。AUTOSAR サーバーインターフェースを要求する SWC の R ポートは、サービスを提供する P ポートに対するクライアント／サーバー呼び出しを行うことにより、インターフェース内に定義されている任意のオペレーションを個別に呼び出すことができます。一方、クライアント／サーバーインターフェースを提供する P ポートは、オペレーションの実装コードを提供します。

クライアント／サーバー通信においてサポートされているのは、複数のクライアントから 1 つのサーバーを呼び出す "n:1"（ただし $n \geqq 0$）という通信形態です。1 つのクライアントが 1 つのリクエストだけで複数のサーバーを呼び出すことはできないため、そのような場合は複数のリクエストが必要です。

4.3.2.3 適合パラメータインターフェース

適合パラメータインターフェースは、適合コンポーネントとの通信に使用されるものです。

各適合パラメータインターフェースには複数の適合パラメータを含めることができます。コンポーネントへの AUTOSAR 適合インターフェースが必要なソフトウェアコンポーネントのポートは、そのポートに RTE の API を作成することにより、インターフェース内に定義されたすべてのパラメータに個別にアクセスすることができます。適合コンポーネントは適合インターフェース（つまり適合パラメータの実装情報）を提供するものです。

4.3.3 ランナブルエンティティとタスク

「ランナブルエンティティ」（「ランナブル」とも呼ばれます）は SWC に含まれる一連のコードを指し、ランタイムおいて RTE によりトリガされます（4.2 項参照）。これは INTECRIO における「プロセス」にほぼ相当します。

各ソフトウェアコンポーネントは 1 つまたは複数のランナブルエンティティで構成され、RTE は、ランタイムにおいて各ランナブルエンティティにアクセスします。ランナブルエンティティは、以下のイベントによりトリガされます。

- タイミングイベント ー周期的スケジューリングイベント（周期的タイマティックなど）です。ランナブルエンティティは通常実行用のエントリポイントを提供します。
- R ポートのデータ受信によりトリガされるイベント（DataReceive イベント）。

AUTOSAR ランナブルエンティティはいくつかのカテゴリに分類されますが、INTECRIO は「カテゴリ 1」のランナブルエンティティをサポートしています。

ランナブルエンティティを実行するためには、そのランナブルエンティティを AUTOSAR オペレーティングシステムのタスクに割り当てます。

ランナブル間変数

RTE においては、1 つのソフトウェアコンポーネント内の複数のランナブルエンティティが「ランナブル間変数」を用いて互いに通信することができます。
INTECRIO ではこれらのランナブル間変数を実験中に測定することができます。

4.3.4 ランタイム環境

RTE（ランタイム環境）については 4.2 項で説明しています。

クロスバー（3.5.5 項）と同様の役割を持つ RTE は、INTECRIO 製品に含まれています。INTECRIO において SWC をソフトウェアシステムやグラフィカルコネクション、OS コンフィギュレーションに割り当てることにより、自動的に RTE の設定が行われます。

5 INTECRIO のコンポーネント

INTECRIO は、一連の基本コンポーネントと Connectivity（接続機能）パッケージで構成されています。

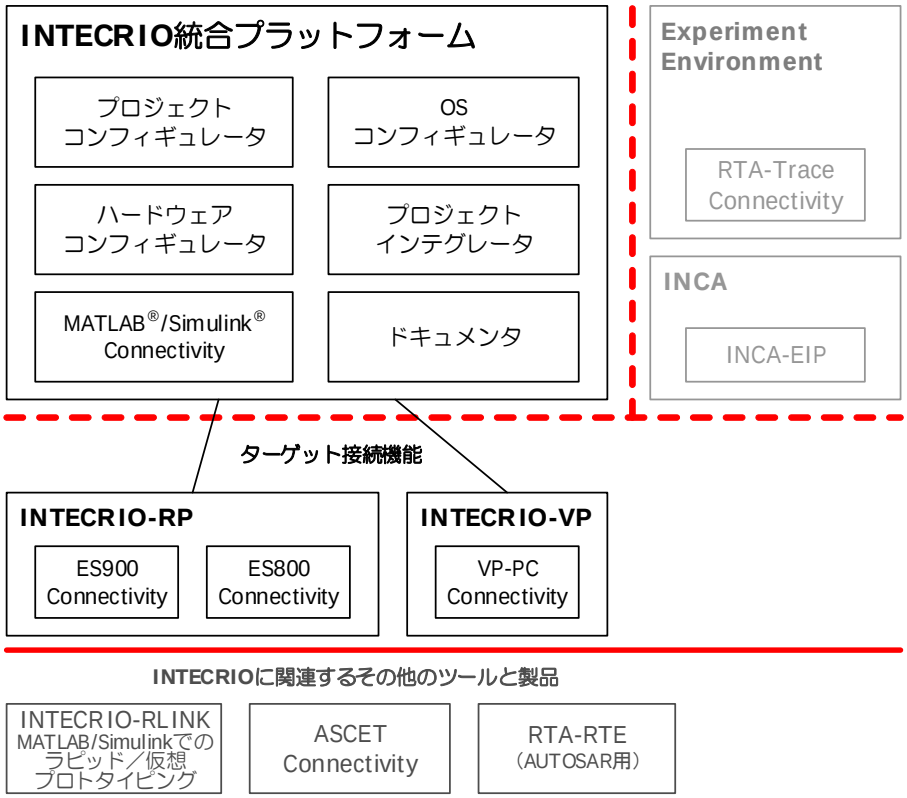


図 5-1 INTECRIO のコンポーネント
黒：INTECRIO
濃い灰色：INTECRIO に関連する ETAS ツール
薄い灰色：INTECRIO での実験に使用できる ETAS ツール

INTECRIO で作業を行う際に不可欠なコンポーネントである「プロジェクトコンフィギュレータ」（5.7 項参照）、「OS コンフィギュレータ」（5.8 項参照）、「ハードウェアコンフィギュレータ」（5.3 項参照）、「プロジェクトインテグレータ」（5.9 項参照）は、INTECRIO 統合プラットフォームに含まれています。また「ドキュメンタ」（5.11 項参照）も INTECRIO 統合プラットフォームに含まれています。

「RTA-TRACE Connectivity」（5.12 項参照）を含む「ETAS Experiment Environment」（5.10 項参照）、「INCA」、「INCA-EIP」はそれぞれ独立した製品です。ETAS Experiment Environment は INTECRIO 製品パッケージに含まれていますが、INCA と INCA-EIP は別途ご購入いただく必要があります。

各ターゲットの Connectivity

「INTECRIO-RP パッケージ」にはラピッドプロトタイピングハードウェア ES900（5.4 項）、ES800（5.5 項）への接続機能が含まれ、「INTECRIO-VP パッケージ」には PC への接続機能（5.6 項）のほか仮想プロトタイピングに必要なものがすべて含まれています。

各 BMT の Connectivity

アプリケーションソフトウェアの機能は各種 BMT（挙動モデリングツール）で記述し、作成したモデルをシステムプロジェクトに統合します。モデル自体の内容の確認や編集は INTECRIO では行えないので、INTECRIO はここでは「統合ツール」としてのみ機能します。

- 「MATLAB®/Simulink® Connectivity」（5.1 項参照）は、INTECRIO 統合プラットフォームに含まれています。
- ASCET Connectivity （5.2 項参照）

  ASCET V6.3 からは、この機能は ASCET-MD に統合されました。

  ASCET V6.2 までは、この機能は ASCET のアドオン「INTECRIO-ASC」として提供されていました。ASCET のインストールメディアから必要に応じてインストールすることができ、ライセンスは INTECRIO 統合プラットフォーム用ライセンスに含まれています。

その他のツールと製品

「INTECRIO-RLINK」を用いると、MATLAB/Simulink 上でラピッドプロトタイピングや仮想プロトタイピングの実験を作成することができます。
「INTECRIO-RLINK」は別途ご購入いただく必要があります。

「RTA-RTE」ツール（4.2 項参照）は AUTOSAR のランタイム環境を提供するものです。

5.1 MATLAB®/Simulink® Connectivity

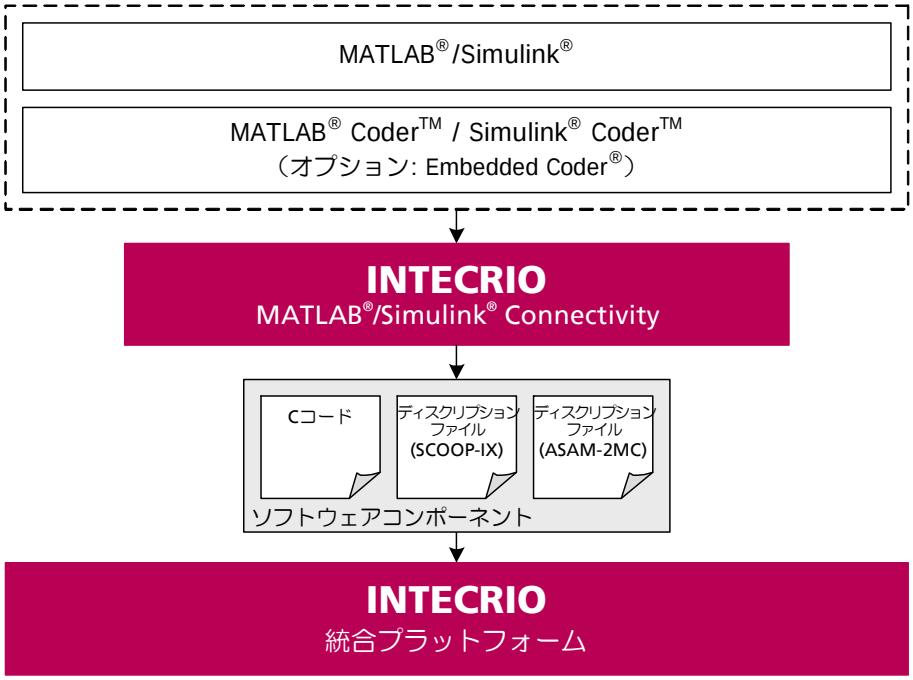


図 5-2 INTECRIO-RP：MATLAB®/Simulink® Connectivity

INTECRIO 統合プラットフォームには「MATLAB®/Simulink®Connectivity」が含まれていて、Simulink モデルを INTECRIO 環境に統合してラピッドプロトタイピングを行うための機能がすべてサポートされています。専用のアドオンは必要はありません。

この機能は、MATLAB® Coder™ + Simulink® Coder™ （+ オプション : Embedded Coder®）で生成されたコードの統合を可能にします。

サポートされている MATLAB/Simulink のバージョンは、R2016a ～ R2021b、および INTECRIO V5.0 のリリース時点においてリリースされているそれらのサービスパックです。有効な複数のバージョンがインストールされている場合は、それらがすべて同時にサポートされます。

サポートされていないバージョンの MATLAB/Simulink では、INTECRIO で使用可能なモデルコードを生成することはできません。

MATLAB/Simulink R2006b ～ R2015b で生成されたモデルコードは、INTECRIO V5.0 にインポートして統合することはできます。

MATLAB/Simulink Connectivity がインストールされる際、MATLAB/Simulink のインストール状態が調整され、INTECRIO とのやりとりが行えるようになります。

MATLAB/Simulink Connectivity には以下の内容が含まれます。

- INTECRIO ターゲット `irt.tlc` が追加されます。Simulink モデルを INTECRIO でさらに処理する場合は、このターゲットを選択します。
- INTECRIO ターゲット `ier.tlc` が追加されます。MATLAB Coder + Simulink Coder + Embedded Coder で生成された Simulink モデルを INTECRIO で使用する場合、このターゲットを選択します。
- 以下のいずれかの環境でのコード生成時に、SCOOP-IX ディスクリプションファイル（第 6 章参照）が自動作成されます。
  - MATLAB Coder + Simulink Coder
  - MATLAB Coder + Simulink Coder + Embedded Coder

  ディスクリプションファイルの内容は 5.1.2 項に説明されています。
- INTECRIO におけるシステムの統合やビルド処理が Simulink から操作できます（「ワンクリックプロトタイピング」）。
- Simulink でモデルを変更した際、そのモデルを INTECRIO に再インポートし、モジュールを更新することができます。

INTECRIO インストール時にインストール済みの MATLAB/Simulink が見つからない場合、または INTECRIO と共に使用する MATLAB/Simulink バージョンを変更したい場合は、以下のように操作してください。

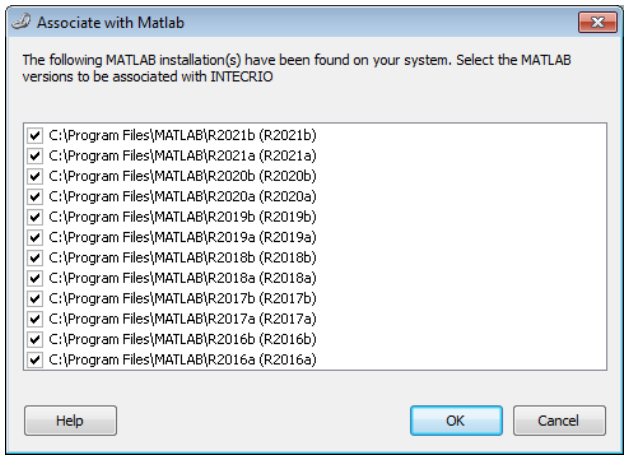
インストール終了後に INTECRIO と MATLAB/Simulink を関連付ける

> **注記**
>
> ネットワークドライブ上にインストールされた MATLAB/Simulink を使用する場合は、**Help** ボタンをクリックし、表示されるメッセージに従って操作してください。

1. 所定の手順で MATLAB/Simulink をインストールします。

2. Windows スタートメニューの INTECRIO フォルダから **Connectivity > Associate with Matlab** を選択します。

   “Associate with Matlab” ウィンドウが開き、サポートされている MATLAB/Simulink のバージョンのうち PC 上にインストールされているものが表示されます。



3. “Associate with Matlab” ウィンドウで、MATLAB/Simulink バージョン（複数可）を選択します。
4. **OK** をクリックします。

   選択された MATLAB/Simulink バージョンが INTECRIO に関連付けられます。

5.1.1 Simulink モデル作成時の注意点

Simulink モデルを INTECRIO で使用するには、モデルの作成時に注意すべき点がいくつかありますが、ここではその中で最も重要な点を説明します。Simulink でのモデリングについての一般情報は、7.2 項「Simulink によるモデリング」を参照してください。

- INTECRIO でシグナルソースまたはシグナルシンクとして認識されるのは、モデルの最上位レベルのポートだけです。

  注記

  下位レベルのポートは、シグナルソースやシグナルシンクとして認識されません（ただし MATLAB/Simulink オンラインヘルプビューアの INTECRIO セクションに例外事項が示されています）。

- INTECRIO は、MATLAB Coder + Simulink Coder（+ Embedded Coder）で生成された浮動小数点コードをサポートします。

  つまり INTECRIO は、MATLAB Coder + Simulink Coder でコード生成できるすべての Simulink ブロックをサポートしています。
- コード生成時に SCOOP-IX ファイルが作成されるように、ターゲットとして INTECRIO ターゲット `irt.tlc` または `ier.tlc` を選択する必要があります。
- INTECRIO では複数の Simulink モデルを統合することができますが、この際、各モデルにそれぞれ異なる固定ステップ幅の統合アルゴリズム（連続ステート用ソルバ）を割り当てることができます。

- INTECRIO（V4.7.2 以降）は、Simulink のモデル参照機能をサポートしています。

5.1.2 ディスクリプションファイルの内容

コード生成時において自動生成される「SCOOP-IX」というインターフェースディスクリプションファイルには、以下のような情報のうち、Simulink 内で実際に定義されているものについての記述が出力されます。

- シグナルソースとシグナルシンク（モデルの最上位レベルの入力ポートと出力ポート）
  - 名前
  - 物理値の範囲
  - 実装データの型と値の範囲（物理値の範囲と量子化の式により算出することもできます。）
  - 量子化の式
- Simulink の一次元信号に対応するシグナル
- Simulink の複素数型信号の実数部を表すシグナルと、虚数部を表すシグナル
- パラメータと変数（次元数の区別はありません。）
  - 名前
  - 物理値の範囲
  - 実装データの型と量子化
  - 階層情報、つまりモデル内における正確な位置
- 起動インターフェース（プロセス）
  - タイミング情報（周期的プロセスの場合）

5.2 ASCET Connectivity

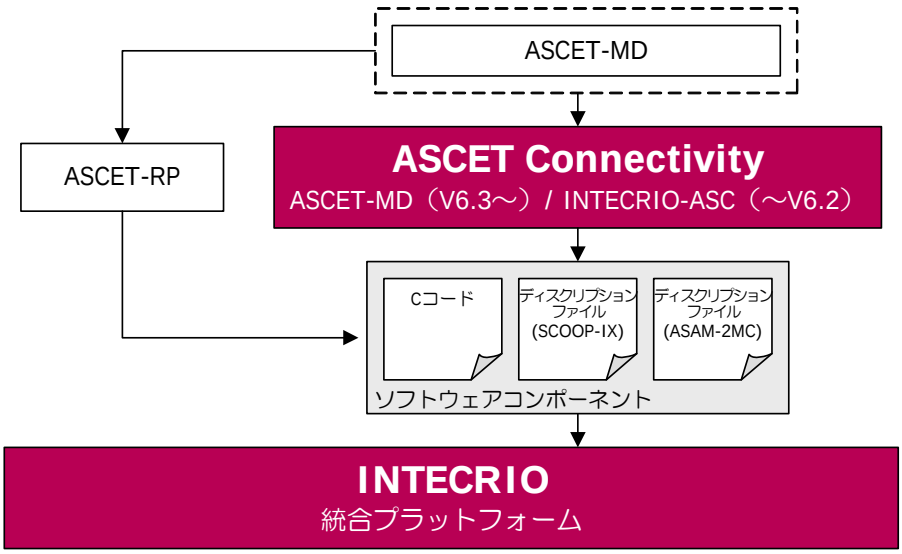


図 5-3 ASCET Connectivity

ASCET Connectivity には、INTECRIO で ASCET モデルを統合してラピッドプロトタイピングを行うために必要なすべての要素が含まれています。

> **注記**
>
> V5.0.2 以降の INTECRIO は、ES1000 および RTPRO-PC ハードウェアシステムをサポートしていません。これらのハードウェアシステムを含む ASCET モデルを INTECRIO V5.0.2 に移行することはできません。

ASCET V6.2 までは、この機能は ASCET のアドオン「INTECRIO-ASC」として提供されていました。INTECRIO-ASC をインストールすると、インストール済みの ASCET にファイルが追加され、必要な設定が行われます。INTECRIO-ASC は ASCET V5.1 ～ V6.2 に対応しているため、複数バージョンの ASCET がインストールされている場合は、それらがすべてサポートされます。

ASCET V6.3 から ASCET Connectivity は ASCET-MD に統合され、アドオンのインストールは必要なくなりました。

ASCET Connectivity には以下の機能が含まれます。

- ターゲット （`Prototyping`、 `ES900`） が追加されます。ASCET モデルを INTECRIO で扱うには、いずれか 1 つのターゲットを選択してコードを生成します。

  > **注記**
  >
  > ASCET connectivity は ES8xx ハードウェアをサポートしていません。

- ASCET のプロジェクトエディタで、INTECRIO をラピッドプロトタイピング環境として選択できます。
- ASCET プロジェクトのコード生成時に、INTECRIO でのラピッドプロトタイピング実験に必要なすべてのファイル（C コード、ASAM-MCD-2MC ファイル、SCOOP-IX ディスクリプションファイル）が生成されます。

  SCOOP-IX ディスクリプションファイルに出力される内容については 5.2.2 項を参照してください。

  さらに、OS コンフィギュレーションが記述された `*.oil` ファイルが生成されます。このファイルはマニュアル操作で INTECRIO にインポートします。
- INTECRIO 用にコード生成を行うことにより、プロジェクトを INTECRIO に自動的にインポートできるようにします。
- 未解決のグローバル変数とメッセージの扱いを、INTECRIO に合わせて調整します。
- 既存のプロジェクトを INTECRIO 用に移行できます。
- INTECRIO におけるシステムの統合やビルド処理を、ASCET から操作すること（「ワンクリックプロトタイピング」）ができます。
- ASCET でモデルを変更した際、そのモデルを再び INTECRIO に自動的にインポートできます。

5.2.1 ASCET モデル作成時の注意点

INTECRIO で使用する ASCET モデルを作成する際は ASCET のすべての機能を利用できますが、以下のような注意が必要です。

- 未接続のメッセージ（対応する送信メッセージが存在しない受信メッセージと、対応する受信メッセージが存在しない送信メッセージ）は、INTECRIO 内でそれぞれ「シグナルシンク」および「シグナルソース」として扱われます。

  また必要に応じて、接続済みのメッセージをシグナルシンクやシグナルソースとして使用することもできます。
- 未解決のメッセージ（対応するエクスポートメッセージが存在しないインポートメッセージ）がプロジェクト内にある場合、INTECRIO 用のコード生成時にエラーメッセージが発行されます。

  この場合は、未解決のメッセージを自動的に解決するか、またはマニュアル操作でメッセージを解決してください。
- グローバル変数やグローバルパラメータの使用は、不可能ではありませんが、避けてください。
- INTECRIO 用コードを生成するには、プロジェクトオプションで適切なターゲットを選択する必要があります。

5.2.2 ディスクリプションファイルの内容

コード生成時において自動生成される「SCOOP-IX」というインターフェースディスクリプションファイルには、以下のような情報のうち、ASCET 内で実際に定義されているものについての記述が出力されます。

- シグナルソースとシグナルシンク（モデルの最上位レベルの入力ポートと出力ポート）
  - 名前
  - 物理値の範囲
  - 実装データの型と値の範囲（物理値の範囲と量子化の式により算出することもできます。）
  - 量子化の式
- パラメータと変数（次元数の区別はありません。）
  - 名前
  - 物理値の範囲
  - 実装データの型と量子化
  - 階層情報、つまりモデル内における正確な位置
- 起動インターフェース（プロセス）
  - タイミング情報（周期的プロセスの場合）

5.3 ハードウェアコンフィギュレータ

制御アルゴリズムの開発工程において、実際の制御対象の挙動を検証する必要がある場合、「ラピッドプロトタイピングシステム」を用いてそれを実現することができます。ラピッドプロトタイピングシステムでは、適切な I/O 装置を用いて接続される一連のセンサやアクチュエータが制御対象（「テクニカルプロセス」とも呼ばれます）と見なされます。

ここで使用される I/O 装置については、実際の技術プロセスモデルに合わせた設定が必要です。たとえば A/D コンバータであれば、入力範囲、サンプル & ホールド回路などを設定し、さらにシグナルを制御アルゴリズムに合わせてコンディショニングする必要もあります。「コンディショニング」とは、一般的に、測定された電気信号値を物理シグナル値に変換することを指します。

これらの処理は、INTECRIO のハードウェアコンフィギュレータとターゲット Connectivity パッケージ（5.4 項と 5.5 項を参照してください）により行われます。

アルゴリズムが、「仮想モデル」の状態で基本的な妥当性チェックを通過した後は、そのアルゴリズムの詳細なエラーの有無を調べ、機能や品質、さらにはコードサイズの面で要件を満たすように改良し、最適化していきます。この際、トラブルシューティングの 1 つの方法として、ハードウェアコンフィギュレータの機能により、ETAS Experiment Environment においてハードウェアのシグナル値を直接測定することができます。

5.3.1 廃止されたハードウェア

V4.7.3 より、INTECRIO は ES1000 および RTPRO-PC ハードウェアをサポートしていません。

V5.0.0 までの INTECRIO で既存のワークスペースを開くと、これらのハードウェアシステムは表示されますが、ハードウェアシステムとターゲットには赤い X 印が付記され、使用不可であることが示されていました。

INTECRIO V5.0.1 以降、開いたワークスペースに含まれる ES1000 や RTPRO-PC ハードウェアシステムは自動的に削除され、ES800、ES900、VP-PC ハードウェアシステムのみが表示されます。

注記

ES1000 または RTPRO-PC ハードウェアシステムが削除されると、それらを使用していたシステムプロジェクトは、使用できなくなります。

そのようなシステムプロジェクトは、ビルドする前に、新しいハードウェアシステムやデバイスを追加し、ハードウェアとソフトウェアを接続して、OS をセットアップする必要があります。詳細な操作方法は、オンラインヘルプを参照してください。

5.3.2 HWX のインポート／エクスポート

ハードウェアのコンフィギュレーションは、マニュアル操作で設定する（5.4.1 項、5.5.1 項参照）代わりに、ハードウェアディスクリプション（`*.hwx` ファイル）のエクスポート／インポート機能を利用することができます。この操作を行うには、ショートカットメニューからコマンドを選択し、所定のダイアログボックスを開きます。

他のツール[1] で作成された *.hwx ファイルをインポートでき、それらのツール用の *.hwx ファイルをエクスポートすることも可能です。

注記

ハードウェアコンフィギュレーションに、ASAM-MCD-2MC ファイルを必要とするバイパス[a] コンフィギュレーションや *.xml ファイルを必要とするデイジーチェーンコンフィギュレーションが含まれている場合は、以下の点に注意が必要がです。

- 上記のようなハードウェアコンフィギュレーションをエクスポートする際には、所定のファイル（*.a2l ／ *.xml）もエクスポートする必要があります。詳細はオンラインヘルプを参照してください。
- 上記のようなハードウェアコンフィギュレーションを正しくインポートするには、*.xml ファイルと同じ場所に所定のファイル（*.a2l ／ *.xml）が保存されていて、HXW インポータがこれらのファイルを自動的に見つけることができるようになっている必要があります。

ただし ASCET からエクスポートされた ASAM-MCD-2MC ファイルを INTECRIO にインポートすると、エラーが発生する可能性があるため、オリジナルの ASAM-MCD-2MC ファイルを使用することをお勧めします。

a. ETK/XETK/FETK/XCP バイパス

ディスクリプションには 2 種類のフォーマット（HWX1 と HWX2）があり、どちらもファイル拡張子は *.hwx です。ただし HWX1 フォーマットはすでに廃止されています。INTECRIO V5.0 では HWX2[2] しか使用できないので、HWX1 をインポートしようとするとエラーが発行されます。

インポートされたハードウェアディスクリプションは、既存の、または新しく作成されるハードウェアシステム（ES900、ES800、VP-PC）にマッピングされます。

インポート処理の実行状態は、ログウィンドウに表示されます。

*.hwx ファイルに INTECRIO がサポートしていないデバイスが含まれていた場合、それらのアイテムはインポートされず、警告が発行されます。

5.3.3 イーサネットコントローラと「XCP on UDP」

ラピッドプロトタイピング用ターゲット（ES910、ES830）がサポートするイーサネットコントローラは、UDP を用いた XCP バイパスと X/FETK バイパスに使用することができます。

イーサネットコントローラは、最大 4 チャンネルの「XCP on UDP」インターフェースと、1 つ（ES910 の場合）または 3 つ（ES830 の場合）の X/FETK バイパスデバイスをサポートしています。

注記

1 つのハードウェアシステム内で使用できる XCP インターフェース（「XCP on UDP」、「XCP on CAN」など）の最大数は、合計で 4 チャンネルです。

1. INTECRIO、ASCET V6.3 以降、ASCET V5.1 ～ V6.2 と INTECRIO-ASC または ASCET-RP V5.3 以降、INTECRIO-RLINK、またはユーザー固有の ASCET 拡張機能
2. HWX2 フォーマットは INTECRIO（V3.2 以降）および ASCET-RP（V6.1 以降）のハードウェアコンフィギュレータで使用されるものです。

図 5-4 に、WS ブラウザに表示されるイーサネットコントローラと「XCP on UDP」を示します。

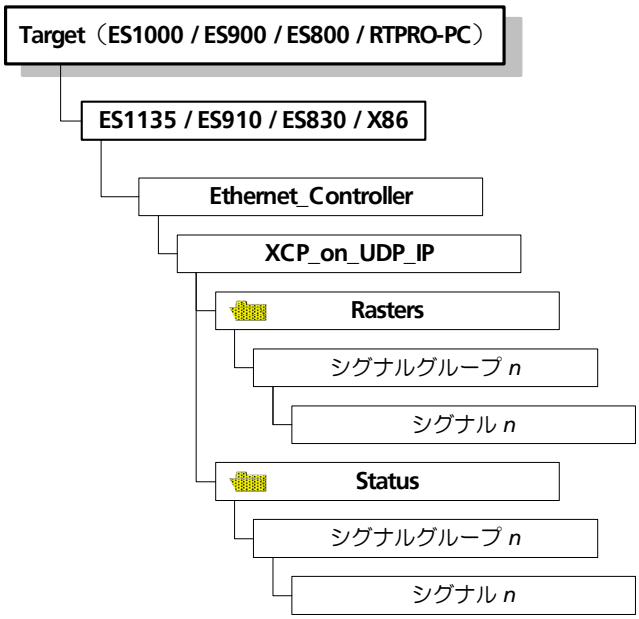


図 5-4 WS ブラウザに表示されるイーサネットおよび「XCP on UDP」のツリー階層

5.3.4 CAN ゲートウェイ

「XXX to CAN Gateway」は、ETK、X/FETK、XCP on CAN、XCP on UDP デバイスから CAN デバイスへのゲートウェイを生成する機能で、両デバイス間の信号を接続するためのゲートウェイを半自動的に生成します。ゲートウェイの挙動は `*.xml` という設定ファイルで定義されます。詳細はオンラインヘルプを参照してください。

5.4 ES900 Connectivity とハードウェアコンフィギュレータ

INTECRIO V5.0 は ES900 ハードウェアシステムをサポートし、ES900 システムで使用できるハードウェアには、ES910.2 ／ ES910.3 ラピッドプロトタイピングモジュール、ES920 モジュール（FlexRay）、ES921 モジュール（CAN ポートの追加）、ES922 モジュール（CAN ／ CAN FD ポートの追加）があります。また ES4xx、ES63x、ES930 もサポートされています。ES900 の統合とコンフィギュレーション設定は、ES900 Connectivity とハードウェアコンフィギュレータの機能を用いて行われます。

注記

ハードウェアコンフィギュレータの ES900 設定機能は、INTECRIO-RP パッケージがインストールされていて、かつユーザーがそのライセンスキーを所有している場合に限り使用できます。

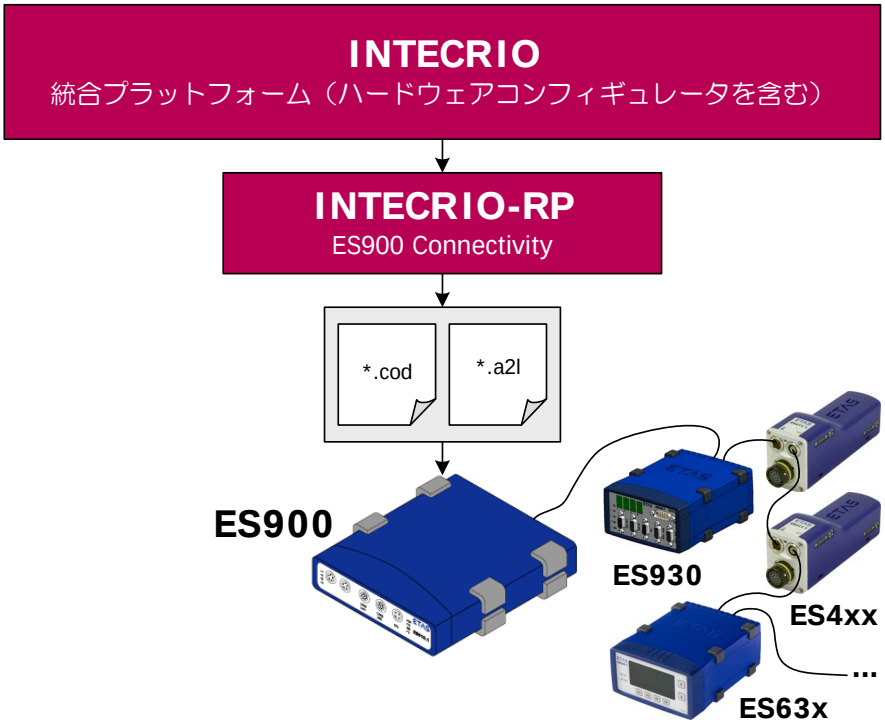


図 5-5 INTECRIO-RP：ES900 Connectivity

ハードウェアコンフィギュレータは、ES900 ハードウェアシステムにモデル入出力用物理シグナルを割り当て、プロジェクトコンフィギュレータ（5.7 項参照）でこれらのシグナルをソフトウェアファンクションモデルに接続します。ハードウェアコンフィギュレータは ES900 の実験用インターフェースを設定するためのものですが、この設定を行う際は、実際のハードウェアは必要ありません。

ES900 に搭載された各インターフェースについては、5.4.2 項または各ハードウェア製品のユーザーマニュアル、または INTECRIO のオンラインヘルプを参照してください。

5.4.1 ハードウェアコンフィギュレータを用いた ES900 のコンフィギュレーション設定

ハードウェアコンフィギュレータには、ES900 システムのコンフィギュレーション設定に必要な以下のコンポーネントが含まれています。

- 汎用的なフレームワーク。エディタや、他の INTECRIO コンポーネント（OS コンフィギュレータ、プロジェクトインテグレータ、ETAS Experiment Environment など）用のインターフェースなどが含まれます。
- 各デバイス（または通信インターフェース）用のエディタ拡張機能。各デバイス用のさまざまなパラメータが複数のタブに分類されて表示され、設定を行うことができます。
- デバイスドライバコンフィギュレーションの生成

これらのコンポーネントは以下の用途に使用されます。

- インターフェースのコンフィギュレーション設定（デイジーチェーン、CAN、FlexRay、LIN コンフィギュレーションのロード）
- OS コンフィギュレータで定義される各タスクへの入出力アクションの割り当て

- 物理シグナル（モデル値）から入出力シグナル（ドライバ値）への変換式の定義

ハードウェアシステムの構成は、WS ブラウザにツリー表示されます。ES900 は最上位レベルの Hardware/Hardware Systems フォルダに位置し、シミュレーションコントローラ（ES910）がそのすぐ下のレベルに配置されます。このコントローラが「マスタ」となり、使用されるデバイスはすべて「スレーブ」として機能します。各デバイスのハードウェアシグナルはさらに下位のレベルに配置されます。各レベルにはそれぞれパラメータがあり、各パラメータの値をエディタで編集することができます。

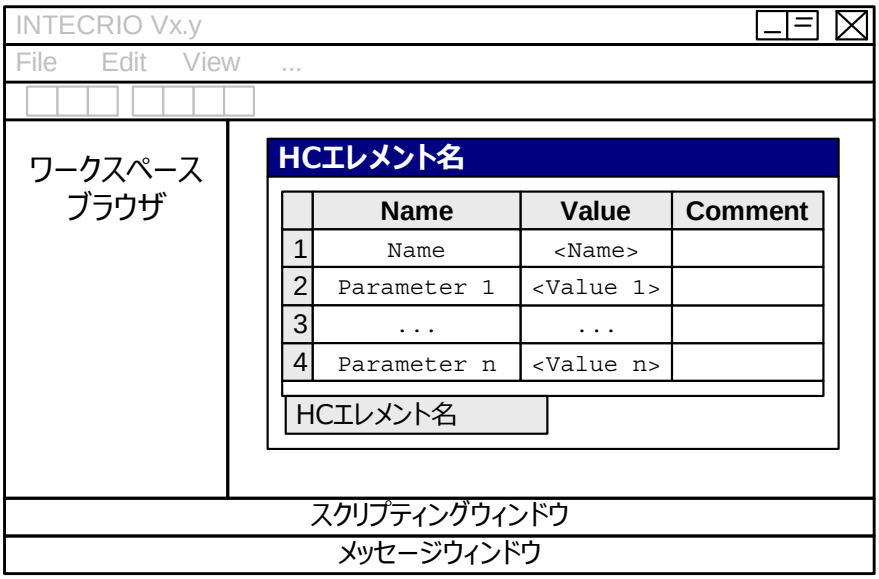


図 5-6 ハードウェアコンフィギュレータ ― ウィンドウ構成

編集の対象となるアイテム（コントローラ、デバイス、シグナルグループ、シグナル）を WS ブラウザで選択してエディタを開くと、そのアイテムに対応したエディタウィンドウが開きます。一度に複数のエディタを開くことができます。

エディタには各アイテムのすべてのパラメータが表形式で表示されます。タブやパラメータの数と名称は、アイテムによって異なります。

マニュアル設定

INTECRIO で使用するハードウェアシステムの構成とパラメータ設定は、オフライン状態で定義することができます。ただしその際には以下のルールが適用されます。

- 任意の数の ES900 システムを作成することができ、システムごとに必ず 1 つのターゲット（ES910）を含めます。
- 1 つのハードウェアシステムは、マスタであるターゲットと、マスタが制御する 1 つまたは複数のスレーブインターフェース（デバイス）で構成されます。
- 各デバイスは、すべて同じレベルに配置します。デバイスのさらに下のレベルにデバイスを組み込むことはできません。
- 各デバイスはマスタに割り当てられます。

- デバイスによっては、1 つの ES900 システム内に同じものを複数個割り当てることができます。割り当て可能なデバイスの数は表 5-1 のとおりです。

デバイス	ES910	注記
CAN_Controller	0 ～ 2 (4)	ES921 で 2 つの CAN I/F を追加可能 ES922 で 2 つの CAN ／ CAN FD I/F を追加可能
CAN_Controller / XCP_on_CAN	0 ～ 4	システム内で使用できる XCP インターフェース（XCP_on_CAN と XCP_on_UDP）の合計最大数は 4 です。
LIN	0 ～ 2	LIN インターフェースは、メインの 2 つの CAN インターフェースと同じ EC910 ハードウェア接続を使用します。 適切なケーブルを使用することにより CAN と LIN を同時に使用できます。
ETK_Bypass	0 ～ 1	フックベースバイパスまたはサービスベースバイパスに使用できます。
SystemDevice (ES910)	0 ～ 1	
ES920 (FlexRay)	0 ～ 1	
DaisyChain（I/O インターフェース）	0 ～ 1	I/O インターフェースにデイジーチェーン接続された ES4xx ／ ES63x ／ ES930 モジュール
Ethernet_Controller / XCP_on_UDP	0 ～ 4	詳しくは 58 ページの 5.3.3 項を参照してください。 システム内で使用できる XCP インターフェース（XCP_on_CAN と XCP_on_UDP）の合計最大数は 4 です。
Ethernet_Controller / X/FETK_Bypass	0 ～ 1	フックベースバイパスまたはサービスベースバイパスに使用できます。 X/FETK バイパスデバイスは XETK 用に使用できます。

注記

INTECRIO では、ES900 システムに ETK と XETK を接続して使用することができます。XETK バイパスデバイスは、ハードウェアコンフィギュレーション内の Ethernet コントローラに追加しますが、XCP バイパス経由で XETK にアクセスすることもできます。

XETK ハードウェアは、ES910 の **ECU** ポートに接続します。

INTECRIO V5.0.1 以降において、ES910 での FETK バイパスは推奨されません。FETK バイパスには ES830 と ES89x の組み合わせを使用してください。

XETK のコンフィギュレーション設定については INTECRIO のオンラインヘルプを参照してください。

表 5-1 1 つのシステムコントローラに割り当て可能なデバイスの数

- デバイスのうち、CAN ／ CAN FD インターフェースについては、各インターフェースにユニークな ID を割り当てる必要があります。

  その他のインターフェースの場合、ID は自動的に割り当てられます。

  いずれの場合も、使用するボードのアドレスが正しく設定されていることをハードウェアコンフィギュレータで確認してください。

- CAN FD を使用するには、ES922 を ES910 に組み込み、ES922 のポート（CAN* （ES922 FD））を CAN コントローラに割り当てます。
- さまざまなタイプのインターフェースをマスタとして割り当てることができます。

> **注記**
>
> アイテムを追加する際、ハードウェアコンフィギュレータは、ツリービュー内で現在選択されているレベルに追加できるアイテムのみを、選択肢として提示します。

これらの点を考慮したうえで、インターフェースやデバイスなどのサブレベルアイテムを任意に挿入したり削除したりできます。

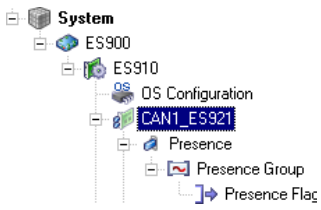
ハードウェアシステムに組み込まれたデバイス（通信インターフェースなど）、またそれらに属するシグナルグループやシグナルの設定も、ハードウェアコンフィギュレータの機能を用いて行います（デイジーチェーンの場合に限り、専用のコンフィギュレーションツールを使用します）。設定時には、以下の点を必ず守ってください。

- 各デバイス（通信インターフェースなど）のパラメータはデフォルト設定されていますが、そのデバイスのシグナルグループやシグナルに関連付けられるタスクやプロセスは OS コンフィギュレータで定義します。1 つのシグナルグループに含まれるシグナルはすべて同時に処理されます。
- シグナルごとに、そのシグナルに接続されるセンサ／アクチュエータ用の変換式を定義できます。以下のような式が使用可能です。
  - 恒等式：`f(phys) = phys`
  - 線形式：`f(phys) = a*phys + b`
- CAN シグナルについては実装情報（変換式と有効値の範囲）を定義できます。
- プラグインモジュールの ES920（FlexRay）と ES921 ／ ES922（CAN ／ CAN FD の追加）は、実際にはいずれか 1 つしか ES910 に組み込めませんが、ここでは同じシステム内に割り当てておくことができます。

  実行時においては、実際に組み込まれているモジュールが有効となり、その他のモジュールについてはエラーが発行されます。

  このような場合は、ES910 のパラメータ “I/O Failure Behavior” を `continue` に設定し、エラーが発生しても実験が停止しないようにしておくことをお勧めします。このように設定されていないと、エラーによって実験が中断されてしまいます。)

  RP モデルの場合は、Presence フラグにより、各プラグインモジュールが実際に使用可能な状態であるかどうかをチェックできます。



- 最大 4 個の CAN コントローラと 2 個の LIN コントローラを同じハードウェアシステムに含めることができます。ただしランタイムにおいてこれらすべてにアクセスするには、適切なケーブルが必要です。

ハードウェアシステムのパーツは、**Copy** ／ **Paste** コマンドでコピーまたは移動することができます。この際、コピー先の設定は上書きされますが、コピー先のパーツが選択されていない場合は新しいパーツが作成されます。

インポート

ハードウェアのコンフィギュレーションは、マニュアル操作で設定する代わりに、ハードウェアコンフィギュレーション（`*.hwx` ファイル）からインポートすることができます。詳しくは 57 ページ「HWX のインポート／エクスポート」を参照してください。

5.4.2 使用可能なデバイス

表 5-2 は、INTECRIO でサポートされている ES900 用デバイスをカテゴリごとに示したものです。

デバイスカテゴリ	名前	機能
シミュレーション コントローラ	ES910	システムコントローラ
通信インターフェース	CAN_Controller / CAN_IO	CAN I/O インターフェース
	CAN_Controller / XCP_on_CAN	XCP バイパス（CAN 経由）
	ETK_Bypass	ETK インターフェース
	LIN_Controller	LIN インターフェース
	ES920	FlexRay インターフェース
	Ethernet_Controller	XCP バイパス（UDP 経由）／ XETK バイパス インターフェース
I/O インターフェース	Daisychain	I/O インターフェースにデイジーチェーン接続された ES4xx ／ ES63x ／ ES930 モジュール
システム インターフェース	SystemDevice	表示と監視モードの制御

表 5-2 ES900 で使用できるインターフェースのタイプと名前

実際に組み込むことができるインターフェースのタイプと数は、表 5-1 を参照してください。

図 5-7 は WS ブラウザに表示されるシステム階層を示したものです。

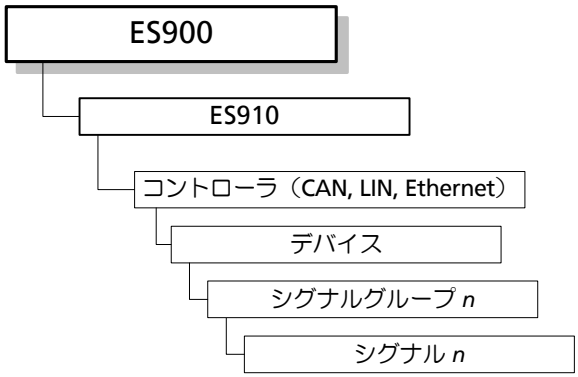


図 5-7 ES900 システムの階層構造

ES910 から制御される各機能は「デバイス」として扱われます。

以下に、各アイテムの機能と、WS ブラウザのツリービューの各レベルに表示される内容を説明します。

シミュレーションコントローラ

シミュレーションコントローラ（図 5-7 の ES910）には以下のものを組み込めます。

- XCP on UDP と XETK 用のイーサーネットインターフェース（`Ethernet`）
- 車載バスインターフェース（CAN ／ CAN FD、LIN、FlexRay）
- ETK インターフェース

通信インターフェース

通信インターフェースは「デバイス」として ES910 コントローラに組み込まれ、ネットワーク（CAN バスなど）上のメッセージ転送を行います。構成に応じて以下のような階層構造になります。

— **CAN コントローラ ＋ CAN I/O** の場合

- 第 1 レベル（ES910 の直下）：CAN コントローラ
- 第 2 レベル：CAN ノード（“CAN_IO” デバイス）
- 第 3 レベル：CAN フレーム（シグナルグループ）と CAN シグナルのフォルダ

  `Frames` フォルダ（CAN フレームフォルダ）の下には以下のようなアイテムが含まれます。

  – 第 4 レベル：CAN フレーム（シグナルグループに相当）

  – 第 5 レベル：標準タイプの CAN シグナルと、CAN シグナルを多重化（つまり 1 つのシグナルを多目的に使用する）ためのマルチプレクサ（“multiplexor”）タイプの CAN シグナル

  – マルチプレクサタイプの CAN シグナルの下の第 6 レベル：CAN 多重化グループ（“multiplex group”）

  – CAN 多重化グループの下の第 7 レベル：CAN 多重化シグナル（“multiplexed signal”）

  `Signals` フォルダ（CAN シグナルフォルダ）の下には以下のようなアイテムが含まれます。

  – 第 4 レベル：標準タイプの CAN シグナル

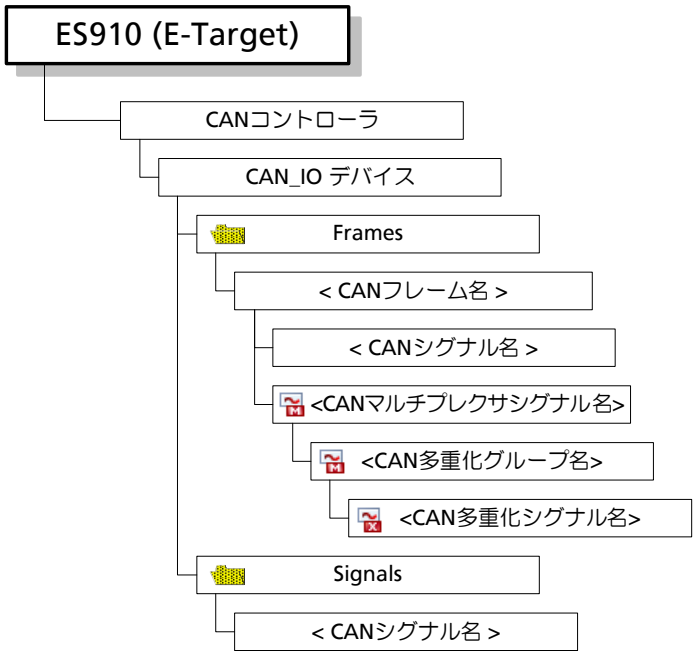


図 5-8 WS ブラウザに表示される CAN I/O インターフェースのツリー階層

— **CAN コントローラ + XCP バイパス** の場合

- 第 1 レベル（ES910 の直下）：CAN コントローラ
- 第 2 レベル：XCP on CAN ノード（“XCP_on_CAN” デバイス）
- 第 3 レベル：ラスタフォルダとステータスのフォルダ

  `Rasters` フォルダ（ラスタフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：CAN フレーム（シグナルグループに相当）
  - 第 5 レベル：標準タイプの CAN シグナルと、マルチプレクサ（“multiplexor”）タイプの CAN シグナル
  - マルチプレクサタイプの CAN シグナルの下の第 6 レベル：CAN 多重化グループ（“multiplex group”）
  - CAN 多重化グループの下の第 7 レベル：CAN 多重化シグナル（“multiplexed signal”）

  `Status` フォルダ（ステータスフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：ステータスシグナルグループ
  - 第 5 レベル：ステータスシグナル

— **ETK インターフェース** の場合

- 第 1 レベル（ES910 の直下）：ETK バイパスデバイス（デバイス “ETK_Bypass”）

> **注記**
>
> ETK インターフェースはフックベースバイパスとサービスベースバイパス（V2 と V3）の両方をサポートしています。両タイプのバイパスを個別に使用したり同時に使用したりできます。
>
> INTECRIO は ETK（8 Mbit/s）のサービスベースバイパスはサポートしていません。

フックベースバイパスの場合、バイパスデバイスの下のレベルは以下のようになります。

- 第 2 レベル：送信用と受信用のシグナルグループ
- 第 3 レベル：シグナル

サービスベースバイパスの場合、バイパスデバイスの下のレベルは以下のようになります。

- 第 2 レベル：送信用と受信用のシグナルグループ、サービスポイント、フックドサービスポイント

  サービスポイントについての情報は ASAM-MCD-2MC ファイルに含まれていますが、バイパスで実際に使用するサービスポイント／フックドサービスポイントの選択とコンフィギュレーション設定は、サービスポイント／フックドサービスポイントの各選択エディタで行います。
- 第 3 レベル（シグナルグループの場合）：シグナル
- 第 3 レベル（サービスポイントの場合）：送信用と受信用のシグナルグループ

  実際に使用するシグナルグループは、サービスポイント／フックドサービスポイントの各選択エディタで設定します。
- 第 4 レベル：シグナル

— **LIN インターフェース** の場合

> **注記**
>
> LIN インターフェースの設定は下記のような構成をマニュアル設定することも可能ですが、LIN ディスクリプションファイルからネットワーク構成をインポートすればより簡単に設定できます。

- 第 1 レベル（ES910 の直下）：LIN コントローラ
- 第 2 レベル：LIN ノード（“LIN I/O” デバイス）

  LIN ノードは、マスタまたはスレーブとして使用します。選択されたノードタイプに応じて、ノードレベルの下のアイテムの有無や方向が異なります。
- 第 3 レベル：スケジュールテーブル用フォルダ（マスタノードの場合のみ）、ステータス表示、フレーム、診断フレーム、シグナル用の各フォルダ

  `Schedule Table` フォルダの下には以下のようなアイテムが含まれます。
  - 第 4 レベル：スケジュールテーブル

- 第 5 レベル：各種フレーム
- 無条件フレーム（"unconditional frames"）の下の第 6 レベル：シグナル
- イベントトリガフレーム（"event-triggered"）および散発フレーム（"sporadic frame"）の下の第 6 レベル：無条件フレームへの参照
- イベントトリガフレームおよび散発フレームの下の第 7 レベル：無条件フレームのシグナルへの参照

`Status` フォルダ（ステータスフォルダ）の下には以下のようなアイテムが含まれます。

- 第 4 レベル：ステータスシグナルグループ
- 第 5 レベル：ステータスシグナル

`Frames` フォルダ（フレームフォルダ）の下には以下のようなアイテムが含まれます。

- 第 4 レベル：イベントトリガフレーム、散発フレーム（マスタノードの場合のみ）、無条件フレームの各フォルダ
- 第 5 レベル：各種タイプのフレーム
- 無条件フレームの下の第 6 レベル：シグナル
- イベントトリガフレームと散発フレームの下の第 6 レベル：無条件フレームへの参照
- イベントトリガフレームと散発フレームの下の第 7 レベル：無条件フレームのシグナルへの参照

`Diagnostic Frames` フォルダ（診断フレームフォルダ）の下には以下のようなアイテムが含まれます。

- 第 4 レベル：診断フレーム（シグナルグループ）
- 第 5 レベル：診断シグナル

`Signals` フォルダ（シグナルフォルダ）の下には以下のようなアイテムが含まれます。

- 第 4 レベル：標準タイプの LIN シグナルが表示されます。

— **FlexRay インターフェース** の場合

- 第 1 レベル（ES910 の直下）：FlexRay IO デバイス（ES920 デバイス）
- 第 2 レベル：ステータス表示、チャンネル、FlexRay フレーム、FlexRay PDU、FlexRay シグナル用の各フォルダ

  チャンネルフォルダ（`Channels` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 3 レベル：2 つのチャンネル
  - 第 4 レベル：スロット（チャンネルごとに最大 2047 スロットを設定可能）
  - 第 5 レベル：フレーム（スロットごとに最大 64 フレームを設定可能）
  - 第 6 レベル：フレームの PDU
  - PDU の下のレベルについては、下記の「PDU フォルダの下には〜」の部分を参照してください。

  ステータスフォルダ（`Status` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 3 レベル：ステータスシグナルグループ

– 第 4 レベル：ステータスシグナル

`Frames` フォルダ（フレームフォルダ）の下には以下のようなアイテムが含まれます。

– 第 3 レベル：FlexRay フレーム
– 第 4 レベル：FlexRay PDU（シグナルグループ）
– 第 5 レベル：標準タイプの FlexRay シグナルとマルチプレクサ（“multiplexor”）タイプの FlexRay シグナル
– マルチプレクサタイプの FlexRay シグナルの下の第 6 レベル：FlexRay 多重化グループ（“multiplex group”）
– FlexRay 多重化グループの下の第 7 レベル：は、FlexRay 多重化シグナル（“multiplexed signal”）が表示されます。

PDU フォルダの下には以下のようなアイテムが含まれます。

– 第 3 レベル：FlexRay PDU（シグナルグループ）が表示されます。
– 第 4 レベル：標準タイプの FlexRay シグナルとマルチプレクサ（“multiplexor”）タイプの FlexRay シグナル
– マルチプレクサタイプの FlexRay シグナルの下の第 5 レベル：FlexRay 多重化グループ（“multiplex group”）
– FlexRay 多重化グループの下の第 6 レベル：FlexRay 多重化シグナル（“multiplexed signal”）

Signals フォルダの下には以下のようなアイテムが含まれます。

– 第 3 レベル：標準タイプの FlexRay シグナル

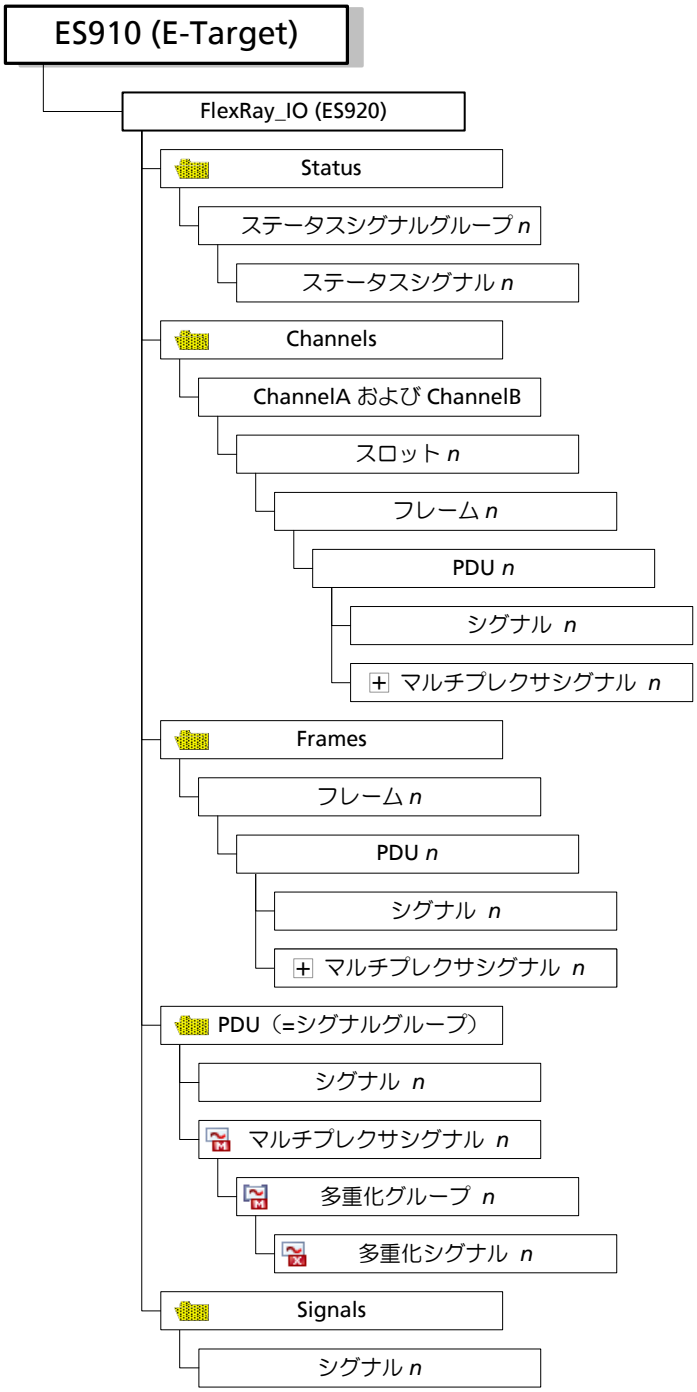


図 5-9 WS ブラウザに表示される FlexRay インターフェースのツリー階層

イーサネットインターフェース

イーサネットインターフェースは、XCP バイパス（XCP on UDP と XETK）または XETK バイパスの設定に使用されます。

XCP バイパス（XCP on UDP と XETK）については、58 ページの 5.3.3 項を参照してください。

— **XETK バイパス** の場合

- 第 1 レベル（ES910 の直下）：イーサネットコントローラ
- 第 2 レベル：X/FETK バイパスデバイス

- 第 3 レベル：サービスポイントとフックドサービスポイント

  サービスポイントについての情報は ASAM-MCD-2MC ファイルに含まれていますが、バイパスで実際に使用するサービスポイント／フックドサービスポイントの選択とコンフィギュレーション設定は、サービスポイント／フックドサービスポイントの各選択エディタで行います。
- 第 4 レベル：送信用と受信用のシグナルグループ

  実際に使用するシグナルグループは、サービスポイント／フックドサービスポイントの各選択エディタで設定します。
- 第 5 レベル：シグナル

システムインターフェース

システムインターフェースは、ES910 の監視モードや表示モードの設定に使用されます。モードごとにシグナルグループが用意されています。

- 第 1 レベル：システムインターフェースデバイス
- 第 2 レベル：各モードに対応するシグナルグループ
- 第 3 レベル：モード用シグナル（各シグナルはデフォルト設定済み）

I/O インターフェース／デイジーチェーン

1 つの ES4xx ／ ES63x ／ ES930 デイジーチェーンを ES910 の背面パネルの **IO** ポートに接続することができます。INTECRIO の WS ブラウザには、このチェーンに含まれる個々のハードウェアモジュールは表示されません。代わりにチェーン全体が 1 つのアイテムとして表示され、各デバイスの各サンプリングレートごとに 1 つのシグナルグループが生成されます。

> **注記**
>
> 他のインターフェースとは異なり、デイジーチェーンのコンフィギュレーションは INTECRIO 内では設定できません。代わりに、外部ツールで作成されたコンフィギュレーションファイルをインポートします。
> コンフィギュレーションファイルを編集した場合は、ショートカットメニューの **Update** コマンドで、その内容を INTECRIO に再インポートしてください。

- 第 1 レベル：デイジーチェーン
- 第 2 レベル：サンプリングレートごとのシグナルグループ
- 第 3 レベル：各シグナルグループに属するシグナル（実際のシグナル数はデイジーチェーンの構成に依存）

5.5 ES800 Connectivity とハードウェアコンフィギュレータ

INTECRIO V5.0 は ES800 ハードウェアシステムをサポートし、ES830 ラピッドプロトタイピングモジュール 1 台と、ECU バスインターフェースモジュール ES891 ／ ES892 ／ ES882 ／ ES886 を最大 4 台使用することができます。

ES800 の統合とコンフィギュレーション設定は、ES800 Connectivity とハードウェアコンフィギュレータの機能を用いて行われます。.

> **注記**
>
> ハードウェアコンフィギュレータの ES800 設定機能は、INTECRIO-RP パッケージがインストールされていて、かつユーザーがそのライセンスキーを所有している場合に限り使用できます。
>
> ES800 ハードウェアシステムを用いた実験を行うには、INCA V7.2.17 以降の INCA と INCA-EIP、または V3.7.7 以降の ETAS Experimental Environment が必要です。

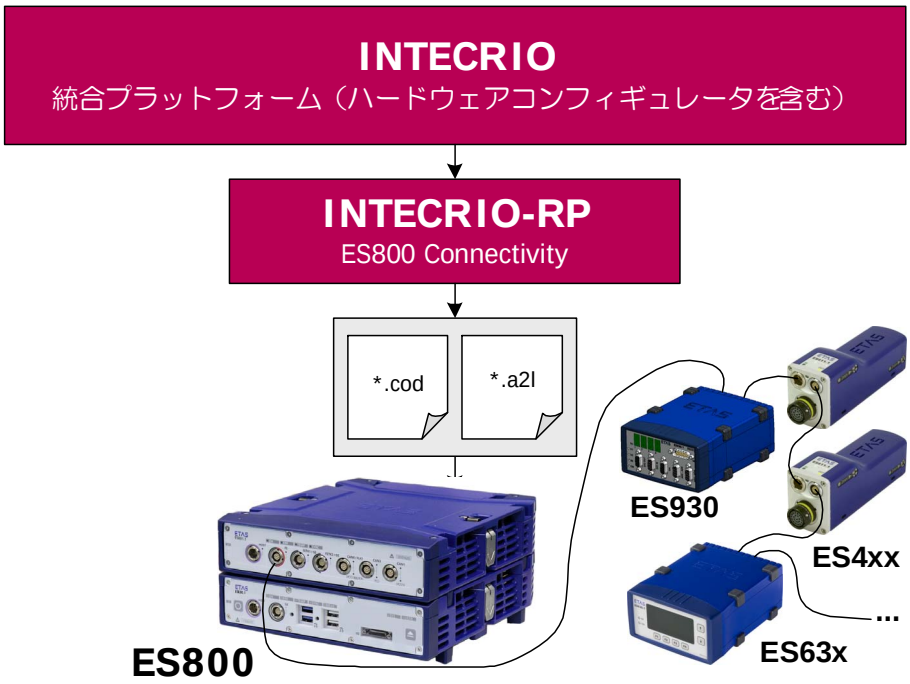


図 5-10 INTECRIO-RP: ES800 Connectivity

ハードウェアコンフィギュレータは、ES800 ハードウェアシステムにモデル入出力用物理シグナルを割り当て、プロジェクトコンフィギュレータ（5.7 項参照）でこれらのシグナルをソフトウェアファンクションモデルに接続します。ハードウェアコンフィギュレータは ES800 の実験用インターフェースを設定するためのものですが、この設定を行う際は、実際のハードウェアは必要ありません。

ES800 に搭載された各インターフェースについては、5.5.2 項または各ハードウェア製品のユーザーマニュアル、または INTECRIO のオンラインヘルプを参照してください。

5.5.1 ハードウェアコンフィギュレータを用いた ES800 のコンフィギュレーション設定

ハードウェアコンフィギュレータには、ES800 システムのコンフィギュレーション設定に必要な以下のコンポーネントが含まれています。

- 汎用的なフレームワーク。エディタや、他の INTECRIO コンポーネント（OS コンフィギュレータ、プロジェクトインテグレータ、ETAS Experiment Environment など）用のインターフェースなどが含まれます。
- 各デバイス（または通信インターフェース）用のエディタ拡張機能。各デバイス用のさまざまなパラメータが複数のタブに分類されて表示され、設定を行うことができます。
- デバイスドライバコンフィギュレーションの生成

これらのコンポーネントは以下の用途に使用されます。

- インターフェースのコンフィギュレーション設定（必要に応じてデイジーチェーン、CAN、FlexRay のコンフィギュレーションをロード）
- OS コンフィギュレータで定義される各タスクへの入出力アクションの割り当て
- 物理シグナル（モデル値）から入出力シグナル（ドライバ値）への変換式の定義

ハードウェアシステムの構成は、WS ブラウザにツリー表示されます。ES800 は最上位レベルの `Hardware/Hardware Systems` フォルダに位置し、シミュレーションコントローラ（ES830）とインターフェースモジュール（ES891 ／ ES892 ／ ES882 ／ ES886）がそのすぐ下のレベルに配置されます。このコントローラが「マスタ」となり、使用されるデバイスはすべて「スレーブ」として機能します。各デバイスのハードウェアシグナルはさらに下位のレベルに配置されます。各レベルにはそれぞれパラメータがあり、各パラメータの値は、ES900 と同様のエディタで編集します（61 ページの図 5-6 参照）。

編集の対象となるアイテム（コントローラ、デバイス、シグナルグループ、シグナル）を WS ブラウザで選択してエディタを開くと、そのアイテムに対応したエディタウィンドウが開きます。一度に複数のエディタを開くことができます。

エディタには各アイテムのすべてのパラメータが表形式で表示されます。タブやパラメータの数と名称は、アイテムによって異なります。

マニュアル設定

INTECRIO で使用するハードウェアシステムの構成とパラメータ設定は、オフライン状態で定義することができます。ただしその際には以下のルールが適用されます。

- 任意の数の ES800 システムを作成することができ、システムごとに必ず 1 つのターゲット（ES910）を含めます。
- 1 つのハードウェアシステムは、マスタである ES830 と、マスタが制御する 1 つまたは複数のスレーブインターフェース（ES891 ／ ES892 ／ ES882 ／ ES886 デバイス）で構成されます。
- 各デバイスは、すべて同じレベルに配置します。デバイスのさらに下のレベルにデバイスを組み込むことはできません。
- 各デバイスはマスタに割り当てられます。
- デバイスによっては、1 つの ES800 システム内に同じものを複数個割り当てることができます。割り当て可能なデバイスの数は表 5-1 のとおりです。

デバイス	ES830	注記
CAN_Controller	0 ～ 5（モジュールごとに）	CAN / CAN FD ES891 ／ ES892 の CAN ポート 4 と 5 を使用するには、ES800 ウェブインターフェースを用いて当該ポート（**CAN4/FLX1** ／ **CAN5/FLX2**）を CAN モードに設定する必要があります。この場合、FlexRay は使用できません。 また、1 つのコネクタを共有する以下のインターフェースを併用するには、所定のケーブル（CBCFI100 など）が必要です。 • CAN1 と CAN2 • ES89x の場合 : – CAN3 と LIN – CAN4 と CAN5 • ES88x の場合 : – CAN3 と CAN4 – CAN5 と LIN
CAN_Controller / XCP_on_CAN	0 ～ 4	使用できるすべての XCP インターフェース（XCP_on_CAN と XCP_on_UDP）の合計は 4 です。
LIN	0 ～ 1（モジュールごとに）	LIN インターフェースは、**CAN3**（ES89x）または **CAN5**（ES88x）と同じコネクタを共有します。 CAN と LIN を併用するには、所定のケーブル（CBCFI100 など）が必要です。
SystemDevice	0 ～ 1	
FlexRay	0 ～ 1（ES89x ごとに）	ES88x と ES892 には FlexRay インターフェースはありません。 ES891 の FlexRay インターフェースを使用するには、ES800 ウェブインターフェースを用いて当該ポート（**CAN4/FLX1** ／ **CAN5/FLX2**）を FlexRay モードに設定する必要があります。この場合、CAN は 3 ポートのみ使用可能です。 また、1 つのコネクタを共有する 2 つのインターフェースを併用するには、所定のケーブル（CBCFI100 など）が必要です。
Ethernet_Controller / Daisychain（I/O インターフェース）	0 ～ 1（モジュールごとに）	Ethernet コントローラにデイジーチェーン接続された ES4xx ／ ES63x ／ ES930 モジュール

デバイス	ES830	注記
Ethernet_Controller / XCP_on_UDP	0 ～ 4	詳細は 58 ページの 5.3.3 項を参照してください。使用できる XCP インターフェース（XCP_on_CAN、XCP_on_UDP）の最大数は、合計 4 つです。
Ethernet_Controller / X_FETK_Bypass	0 ～ 4（表 5-6 参照）	フックベースバイパスまたはサービスベースバイパスに使用できます。X/FETK バイパスデバイスは、モジュールタイプに応じて XETK、BR_XETK、FETK に使用されます。 ES88x: XETK、BR_XETK ES89x: XETK、FETK

注記

INTECRIO では、ES800 システムに XETK、FETK、BR_XETK を接続して使用することができます。X/FETK バイパスデバイスは、ハードウェアコンフィギュレーション内の Ethernet コントローラに追加します。

XETK ハードウェアは ES891 ／ ES892 ／ ES882 ／ ES886 の **FE** ポートに接続します。

BR_XETK ハードウェアは ES882 ／ ES886 の **AE** ポートに接続します。

FETK ハードウェアは ES891 ／ ES892 の **FETK1/GE** ／ **FETK2/GE** ポートに接続し、ES800 ウェブインターフェースを用いてこれらのポートを FETK モードに設定します。

各 ETK の設定についての詳細は、INTECRIO オンラインヘルプを参照してください。

ES800 ウェブインターフェースについての詳細は、『ES800 システムユーザーガイド』を参照してください。

表 5-3　1 つの ES800 システムに割り当て可能なデバイスの数

- CAN インターフェースについては、各インターフェースにユニークな ID を割り当てる必要があります。

  その他のインターフェースの場合、ID は自動的に割り当てられます。

これらの点を考慮したうえで、インターフェースやデバイスなどのサブレベルアイテムを任意に挿入したり削除したりできます。

ハードウェアシステムに組み込まれたデバイス（通信インターフェースなど）、またそれらに属するシグナルグループやシグナルの設定も、ハードウェアコンフィギュレータの機能を用いて行います。設定時には、以下の点を必ず守ってください。

- 各デバイス（通信インターフェースなど）のパラメータはデフォルト設定されていますが、そのデバイスのシグナルグループやシグナルに関連付けられるタスクやプロセスは OS コンフィギュレータで定義します。1 つのシグナルグループに含まれるシグナルはすべて同時に処理されます。
- シグナルごとに、そのシグナルに接続されるセンサ／アクチュエータ用の変換式を定義できます。以下のような式が使用可能です。
  - 恒等式：`f(phys) = phys`
  - 線形式：`f(phys) = a*phys + b`
- CAN シグナルについては実装情報（変換式と有効値の範囲）を定義できます。

- １つの ES89x ／ ES88x モジュールに組み込めるデバイスの最大数は INTECRIO オンラインヘルプに記載されています。ハードウェアシステム内の最大数は、ES800 スタックの構成に応じて異なります。

  ランタイムにすべてのコントローラにアクセスできるようにするには、適切なケーブルが必要です。

ハードウェアシステムのパーツは、**Copy** ／ **Paste** コマンドでコピーまたは移動することができます。この際、コピー先の設定は上書きされますが、コピー先のパーツが選択されていない場合は新しいパーツが作成されます。

インポート

ハードウェアのコンフィギュレーションは、マニュアル操作で設定する代わりに、ハードウェアコンフィギュレーション（`*.hwx` ファイル）からインポートすることができます。詳しくは 57 ページ「HWX のインポート／エクスポート」を参照してください。

5.5.2 使用可能なデバイス

表 5-4 は、INTECRIO でサポートされている ES800 用デバイスをカテゴリごとに示したものです。

デバイスカテゴリ	名前	機能
シミュレーション コントローラ	ES830	
通信インターフェース（ES891 ／ ES892 ／ ES882 ／ ES886）	CAN_Controller / CAN_IO	CAN I/O インターフェース
	CAN_Controller / XCP_on_CAN	XCP バイパス（CAN 経由）
	LIN_Controller	LIN インターフェース
	FlexRay	FlexRay インターフェース（ES891 のみ）
	Ethernet_Controller / X_FETK_Bypass	XETK/BR_XETK/FETK バイパスインターフェース ES88x: XETK、BR_XETK ES89x: XETK、FETK
	Ethernet_Controller / XCP_on_UDP	XCP バイパス（UDP 経由）
I/O インターフェース	Ethernet_Controller / Daisychain	Ethernet コントローラにデイジーチェーン接続された ES4xx ／ ES63x ／ ES930 モジュール
システム インターフェース	SystemDevice	表示と監視モードの制御

表 5-4 ES800 で使用できるインターフェースのタイプと名前

ES830 と ES891 ／ ES892 ／ ES882 ／ ES886 を併用して実際に使用できるインターフェースのタイプと数は、表 5-3 を参照してください。ハードウェアシステム内の最大数は、ES800 スタックの構成に応じて異なります。

図 5-11 は WS ブラウザに表示されるシステム階層を示したものです。

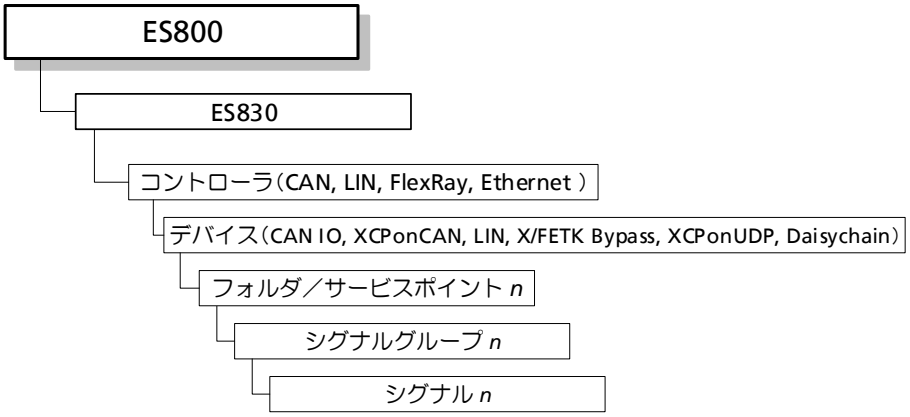


図 5-11 ES800 システムの階層構造

シミュレーションコントローラ

シミュレーションコントローラ ES830 は、最大 4 つのインターフェースモジュール ES891 ／ ES892 ／ ES882 ／ ES886 に接続できます。これらのインターフェースモジュールは、WS ブラウザにおいて独立したノードとしては表示されません。

通信インターフェース

ES891 ／ ES892 ／ ES882 ／ ES886 には以下のインターフェースが搭載されています。

インターフェース	モジュール
車両 CAN バス用インターフェース x 5ch	ES89x, ES88x
車両 FlexRay バス用インターフェース x 1ch （CAN インターフェース x 2ch を置き換える）	ES891
車両 LIN バス用インターフェース x 1ch	ES89x, ES88x
XETK またはデイジーチェーンに使用できるファーストイーサネット x 1ch	ES89x, ES88x
FETK 用ギガビットイーサネット x 2ch	ES89x
BR_XETK 用車載イーサネット x 3ch（ES882）、x 4ch（ES886）	ES88x

表 5-5 ES89x/ES88x デバイスに搭載された通信インターフェース（ハードウェアシステム内の最大数は、ES800 スタックの構成に応じて異なります）

INTECRIO では、ES800 ハードウェアシステムにおいて、ES89x ／ ES88x モジュールごとに各種 ETK を以下の組み合わせで使用することができます。

モジュール	使用可能な ETK タイプと台数
ES891/ES892	FETK（1 ～ 2） XETK（1） FETK（1 ～ 2）と XETK（1）
ES882/ES886	BR_XETK ／ XETK（合計で 1 ～ 3）

注記

1 つの ES800 ハードウェアシステム内で使用できる X/FETK デバイスの最大数は、4 です。

表 5-6 ES89x ／ ES88x モジュールで使用できる 各種 ETK の最大数

ES891 の 2 つのポートは、CAN ポートまたは FlexRay ポートとして使用でき、INTECRIO では、1 つの ES800 ハードウェアシステム内で FlexRay コントローラを 1 つだけ使用することができます。

— **CAN コントローラ ＋ CAN I/O** の場合

- 第 1 レベル（ES830 の直下）：CAN コントローラ
- 第 2 レベル：CAN ノード（“CAN_IO” デバイス）が表示されます。
- 第 3 レベル：CAN フレーム（シグナルグループ）と CAN シグナルのフォルダ

  CAN フレームフォルダ（`Frames` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 4 レベル：CAN フレーム（シグナルグループに相当）
  - 第 5 レベル：標準タイプの CAN シグナルと、CAN シグナルを多重化（つまり 1 つのシグナルを多目的に使用する）ためのマルチプレクサ（“multiplexor”）タイプの CAN シグナル
  - マルチプレクサタイプの CAN シグナルの下の第 6 レベル：CAN 多重化グループ（“multiplex group”）
  - CAN 多重化グループの下の第 7 レベル：CAN 多重化シグナル（“multiplexed signal”）

  `Signals` フォルダ（CAN シグナルフォルダ）の下には以下のようなアイテムが含まれます。

  - 第 4 レベル：標準タイプの CAN シグナル

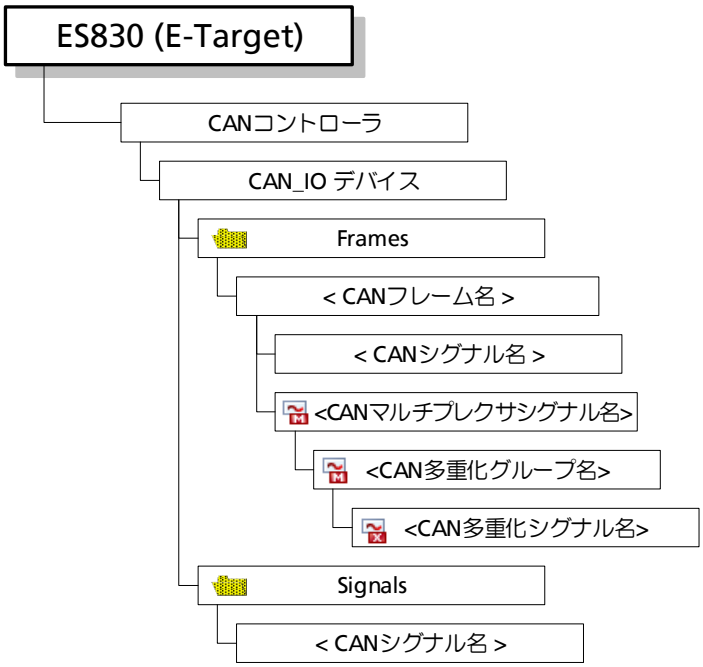


図 5-12 WS ブラウザに表示される ES800 の CAN I/O インターフェースのツリー階層

— **CAN コントローラ + XCP バイパス** の場合

- 第 1 レベル（ES830 の直下）：CAN コントローラ
- 第 2 レベル：XCP on CAN ノード（“XCP_on_CAN” デバイス）
- 第 3 レベル：ラスタフォルダとステータスのフォルダ

  `Rasters` フォルダ（ラスタフォルダ）の下には以下のようなアイテムが含まれます。

  - 第 4 レベル：CAN フレーム（シグナルグループに相当）
  - 第 5 レベル：標準タイプの CAN シグナルと、マルチプレクサ（“multiplexor”）タイプの CAN シグナル
  - マルチプレクサタイプの CAN シグナルの下の第 6 レベル：CAN 多重化グループ（“multiplex group”）
  - CAN 多重化グループの下の第 7 レベル：CAN 多重化シグナル（“multiplexed signal”）

  `Status` フォルダ（ステータスフォルダ）の下には以下のようなアイテムが含まれます。

  - 第 4 レベル：ステータスシグナルグループ
  - 第 5 レベル：ステータスシグナル

— **LIN インターフェース** の場合

注記

LIN インターフェースの設定は下記のような構成をマニュアル設定することも可能ですが、LIN ディスクリプションファイルからネットワーク構成をインポートすればより簡単に設定できます。

- 第 1 レベル（ES910 の直下）：LIN コントローラ

- 第 2 レベル：LIN ノード（“LIN I/O” デバイス）

  LIN ノードは、マスタまたはスレーブとして使用します。選択されたノードタイプに応じて、ノードレベルの下のアイテムの有無や方向が異なります。
- 第 3 レベル：スケジュールテーブル用フォルダ（マスタノードの場合のみ）、ステータス表示、フレーム、診断フレーム、シグナル用の各フォルダ

  `Schedule Table` フォルダの下には以下のようなアイテムが含まれます。
  - 第 4 レベル：スケジュールテーブル
  - 第 5 レベル：各種フレーム
  - 無条件フレーム（"unconditional frames"）の下の第 6 レベル：シグナル
  - イベントトリガフレーム（"event-triggered"）および散発フレーム（"sporadic frame"）の下の第 6 レベル：無条件フレームへの参照
  - イベントトリガフレームおよび散発フレームの下の第 7 レベル：無条件フレームのシグナルへの参照

  `Status` フォルダ（ステータスフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：ステータスシグナルグループ
  - 第 5 レベル：ステータスシグナル

  `Frames` フォルダ（フレームフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：イベントトリガフレーム、散発フレーム（マスタノードの場合のみ）、無条件フレームの各フォルダ
  - 第 5 レベル：各種タイプのフレーム
  - 無条件フレームの下の第 6 レベル：シグナル
  - イベントトリガフレームと散発フレームの下の第 6 レベル：無条件フレームへの参照
  - イベントトリガフレームと散発フレームの下の第 7 レベル：無条件フレームのシグナルへの参照

  `Diagnostic Frames` フォルダ（診断フレームフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：診断フレーム（シグナルグループ）
  - 第 5 レベル：診断シグナル

  `Signals` フォルダ（シグナルフォルダ）の下には以下のようなアイテムが含まれます。
  - 第 4 レベル：標準タイプの LIN シグナルが表示されます。

— **FlexRay インターフェース** の場合

> **注記**
>
> FlexRay インターフェースを搭載しているのは ES891 モジュールのみです。

- 第 1 レベル（ES830 の直下）：FlexRay デバイス

- 第 2 レベル：ステータス表示、チャンネル、FlexRay フレーム、FlexRay PDU、FlexRay シグナル用の各フォルダ

  ステータスフォルダ（`Status` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 3 レベル：ステータスシグナルグループ
  - 第 4 レベル：ステータスシグナル

  チャンネルフォルダ（`Channels` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 3 レベル：2 つのチャンネル
  - 第 4 レベル：スロット（各チャンネルに最大 2047 スロットを設定可能）
  - 第 5 レベル：フレーム（各スロットに最大 64 フレームを設定可能）
  - 第 6 レベル：フレームの PDU
  - PDU の下のレベルについては、下記の「PDU フォルダの下には〜」の部分を参照してください。

  フレームフォルダ（`Frames` フォルダ）の下には以下のようなアイテムが含まれます。

  - 第 3 レベル：FlexRay フレーム
  - 第 4 レベル：FlexRay PDU（シグナルグループ）
  - 第 5 レベル：標準タイプの FlexRay シグナルと**マルチプレクサ**（“multiplexor”）タイプの FlexRay シグナル
  - マルチプレクサタイプの FlexRay シグナルの下の第 6 レベル：FlexRay 多重化グループ（“multiplex group”）
  - FlexRay 多重化グループの下の第 7 レベル：FlexRay 多重化シグナル（“multiplexed signal”）

  PDU フォルダの下には以下のようなアイテムが含まれます。

  - 第 3 レベル：FlexRay PDU（シグナルグループ）
  - 第 4 レベル：標準タイプの FlexRay シグナルとマルチプレクサ（“multiplexor”）タイプの FlexRay シグナル
  - マルチプレクサタイプの FlexRay シグナルの下の第 5 レベル：FlexRay 多重化グループ（“multiplex group”）
  - FlexRay 多重化グループの下の第 6 レベル：FlexRay 多重化シグナル（“multiplexed signal”）

  Signals フォルダの下には以下のようなアイテムが含まれます。

  - 第 3 レベル：標準タイプの FlexRay シグナル

FlexRay インターフェースのツリー階層は、70 ページの図 5-9 を参照してください。

イーサネットインターフェース

ES891 ／ ES892 のギガビットイーサネットインターフェースとファーストイーサネットインターフェースで、X/FETK バイパスの設定が行えます。

ES882 ／ ES886 のファーストイーサネットインターフェースは XETK バイパスに使用でき、車載イーサネットインターフェースは BR_XETK バイパスに使用できます。

> **注記**
>
> ES800 システムでは X/FETK バイパスのみサポートされています。
>
> 各 ES89x/ES88x モジュールで同時に使用できるバイパスデバイスのタイプと最大数は、表 5-6 を参照してください。

XCP バイパス（「XCP on UDP」と X/FETK）については、58 ページの 5.3.3 項を参照してください。

— **X/FETK バイパス** の場合：

- 第 1 レベル（ES830 の直下）：イーサネットコントローラ
- 第 2 レベル：X/FETK バイパスデバイス
- 第 3 レベル：サービスポイントとフックドサービスポイント

  サービスポイントについての情報は ASAM-MCD-2MC ファイルに含まれていますが、バイパスで実際に使用するサービスポイント／フックドサービスポイントの選択とコンフィギュレーション設定は、サービスポイント／フックドサービスポイントの各選択エディタで行います。
- 第 4 レベル：送信用と受信用のシグナルグループ

  実際に使用するシグナルグループは、サービスポイント／フックドサービスポイントの各選択エディタで設定します。
- 第 5 レベル：シグナル

I/O インターフェース／デイジーチェーン

1 つの ES4xx ／ ES63x ／ ES930 デイジーチェーンを ES800 のファーストイーサネットポートに接続することができます。INTECRIO の WS ブラウザには、このチェーンに含まれる個々のハードウェアモジュールは表示されません。代わりにチェーン全体が 1 つのアイテムとして表示され、各デバイスの各サンプリングレートごとに 1 つのシグナルグループが生成されます。

> **注記**
>
> 他のインターフェースとは異なり、デイジーチェーンのコンフィギュレーションは INTECRIO 内では設定できません。代わりに、外部ツールで作成されたコンフィギュレーションファイルをインポートします。
> コンフィギュレーションファイルを編集した場合は、ショートカットメニューの **Update** コマンドで、その内容を INTECRIO に再インポートしてください。

- 第 1 レベル：イーサネットコントローラ
- 第 2 レベル：デイジーチェーンデバイス
- 第 3 レベル：サンプリングレートごとのシグナルグループ
- 第 4 レベル：各シグナルグループに属するシグナル（実際のシグナル数はデイジーチェーンの構成に依存）

システムインターフェース

システムインターフェースは、ES830 の監視モードや表示モードの設定に使用されます。モードごとにシグナルグループが用意されています。

- 第 1 レベル：システムインターフェースデバイス

- 第 2 レベル：各モードに対応するシグナルグループ
- 第 3 レベル：モード用シグナル（各シグナルはデフォルト設定済み）

5.6 PC Connectivity

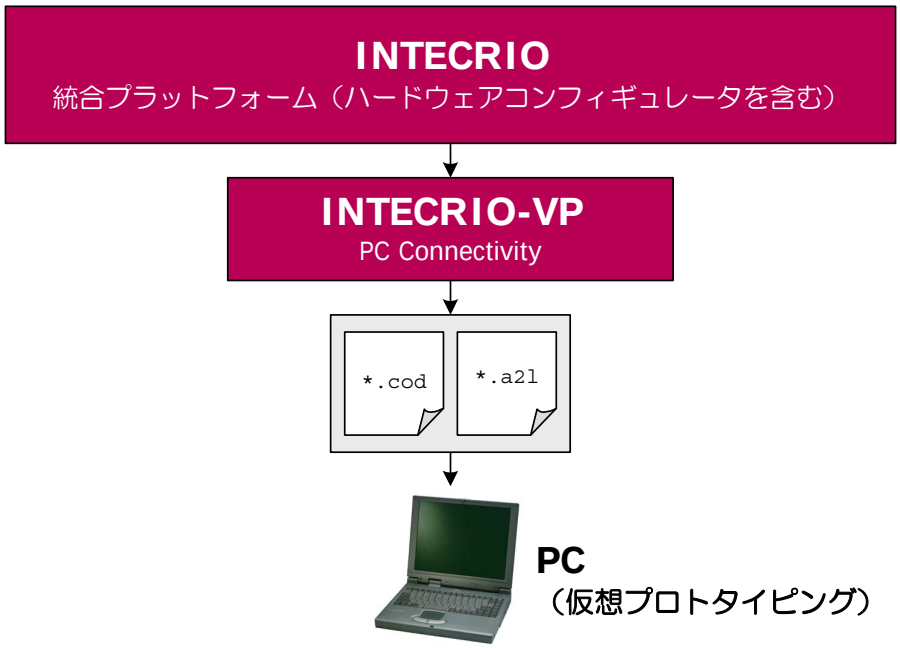


図 5-13 INTECRIO-VP：VP-PC Connectivity

INTECRIO-VP には、INTECRIO を仮想プロトタイピングに用いるための「PC Connectivity」が含まれています。仮想プロトタイピングは、アプリケーションソフトウェアの評価（妥当性の確認）や事前適合に利用できます。V サイクル内の非常に早い段階でソフトウェアの検証や評価を行える、という点が、大きなメリットです。

INTECRIO-VP は、INTECRIO で仮想プロトタイピングを行うために必要な以下のような機能を提供します。

- INTECRIO に VP-PC ターゲットを追加します。このターゲットは、WS ブラウザにおいて他のターゲット（ES900、ES800）と同様に扱われます。
- シミュレーションを任意の速度で（ただし CPU の演算能力とモデルの複雑性の制限内で）実行することができます。
- OSC（OS コンフィギュレータ）により、RTA-OSEK を PC 用オペレーティングシステムとして設定できます（5.8 項参照）。
- 通常のラピッドプロトタイピングの場合と同様に、実行ファイルと ASAM-MCD-2MC ファイルを生成できます（5.9 項参照）。
- 仮想プロトタイピング実験において RTA-TRACE と INCA-EIP を使用でき、Windows タスクバーから VP サービスを利用できます。
- 仮想プロトタイピング実験中に ASCET や MATLAB/Simulink モデルのバックアニメーションを使用できます。

  バックアニメーションは、モデル変数を BMT のディスプレイに表示して、適合変数をモデル内から直接適合することができる機能です。

- 仮想プロトタイピング実験中に Microsoft® Visual Studio® デバッガに接続できます。ただしデバック機能が利用できるのは、ビルド処理と実験が同じ PC 上で行われ、ビルド処理終了後に直ちに実験を開始した場合に限られます。

  仮想 ECU における割り込みをシミュレートするには、仮想マシンがアプリケーションスレッドのスタックを非同期で扱える必要があります。多くの関数が非同期のスタック変更を行うことはできないので、コード行ごとのステップ実行は行えません。

> **注記**
>
> 実際のデバッグ時には、上記のようなステップ実行ではなく、複数のブレイクポイントを設定しておき、ブレイクポイントからブレイクポイントにジャンプしながら実行することをお勧めします。詳しくは PC 用 RTA-OSEK のドキュメントを参照してください。

5.7 プロジェクトコンフィギュレータ

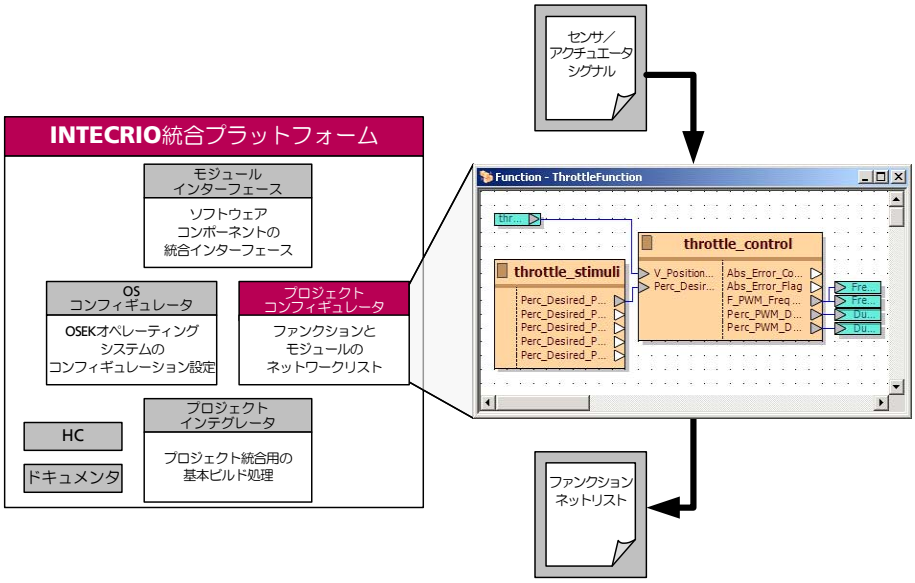


図 5-14　プロジェクトコンフィギュレータ

「プロジェクトコンフィギュレータ」は INTECRIO の統合プラットフォームの一部です。ソフトウェアシステムとシステムプロジェクトの設定に使用されます。プロジェクトコンフィギュレータにはオフラインまたはオンラインで使用できるグラフィックエディタ（図 5-14 の右の部分）が含まれています。

5.7.1 オフラインモード

オフラインモードでは、モジュールと SWC（AUTOSAR ソフトウェアコンポーネント）の表示と、ファンクション、ソフトウェアシステム、システムプロジェクトの作成と設定が行えます。

5.7.1.1 モジュールと SWC

モジュールはワークスペースの `Software/Modules` または `Environment/Modules` フォルダに、SWC はワークスペースの `Software/Modules` フォルダに保存します（31 ページ「モジュールと AUTOSAR ソフトウェアコンポーネント（SWC）」参照）。

プロジェクトコンフィギュレータのグラフィックエディタでは、モジュール／SWC はブロックとして表示され、ブロックの上部にモジュール／ SWC 名が表示されます。

シグナルシンク（入力）はブロックの左側に配置され、シグナルソース（出力）は右側に配置されます。ブロックの色は、そのモジュール作成に使用された BMT を示します（オレンジ - MATLAB/Simulink、緑 - ASCET、赤 - AUTOSAR SWC）。

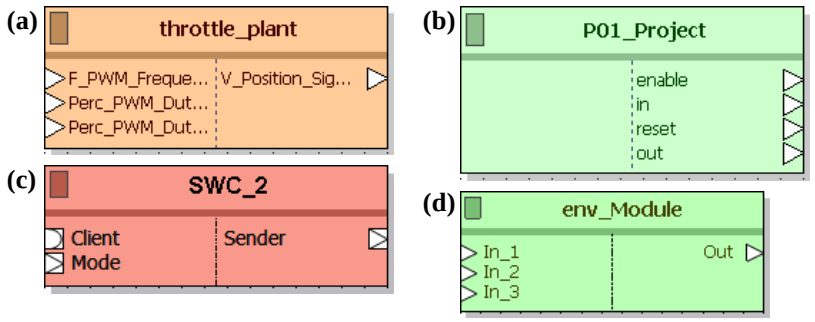


図 5-15 モジュール／ SWC の標準レイアウト
(a): Simulink モジュール、**(b)**: ASCET モジュール、
(c): AUTOSAR SWC、**(d)**: 環境モジュール

ブロックは、初期状態においては、すべてのインターフェースエレメントが見えるような標準サイズで表示されます。シグナルソースとシグナルシンクの名前は一部しか表示されない場合がありますが、マウスカーソルを合わせれば、名前の全体をツールチップに表示することができます。

ブロックの色はオプション設定で変更できますが、同じ BMT で作成されたモジュール／ SWC には必ず同じ色が使用されます。各ブロックのサイズとレイアウトは個別に調整できます。

新しいバージョンの SCOOP-IX ファイルをインポートしてモジュールの内容を更新した後も、これらの表示設定はすべて保持されます。

5.7.1.2 ファンクション

ファンクション（33 ページ「ファンクション」参照）は、プロジェクトコンフィギュレータの機能を用いてワークスペースの `Software/Functions` または `Environment/Functions` フォルダ内に作成し、定義します。

ファンクションのインターフェース（シグナルシンクとシグナルソース）も、プロジェクトコンフィギュレータで作成します。実装情報、シグナルタイプなどは、接続されるエレメントにより決定されます。

ファンクションの外観図はモジュール／ SWC と同様です。シグナルシンクは左側、シグナルソースは右側に配置され、接続されたモジュール／ SWC のポート名が表示されます。ブロックサイズはシグナルシンクとシグナルソースの数に応じて自動的に設定されます。ファンクションのサイズと両側のシグナルシンクやシグナルソースの位置は調整可能です。

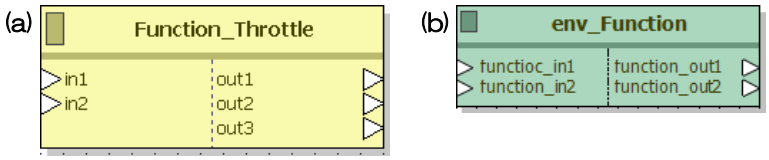


図 5-16 ファンクション **(a)** と環境ファンクション **(b)** の標準レイアウト（外観図）

ファンクションには任意の数のモジュールを割り当てることができますが、ファンクションの中に別のファンクションを割り当てることはできません。また 1 つのファンクションに同じモジュールを 2 回以上割り当てることはできません。だたしグラフィックエディタ上では 1 つのファンクション内に同じモジュールを 2 個以上配置できてしまうため、注意が必要です。

モジュール／ SWC のインターフェースエレメント同士の接続は、グラフィックエディタまたはプロジェクトコンフィギュレータの接続ウィザードで行います。また、モジュールのインターフェースエレメントをファンクションのインターフェースエレメントに接続することもできます。未使用のモジュールインターフェースエレメントはグラフィック表示から削除できます（ただしモジュールから実際に削除されるわけではありません）。接続には以下のルールが適用されます。

- モジュールの場合

  1 つのソース（出力）から複数のシンク（入力）に接続できます。複数のソース（出力）を同一のシンク（入力）に接続することはできません。
- SWC の場合：

  1 つのセンダから複数のレシーバ、また 1 つのサーバーから複数のクライアントに接続できます。スカラ型のセンダとレシーバは、モジュールの入出力に接続可能です。
- 接続されていないソースやシンクがあってもかまいません。
- モジュール間は「静的コネクション」または「動的コネクション」として接続できます。動的コネクションはプログラム実行時にルートを変更することができます。ただし SWC の場合はすべて静的コネクションとなります。

5.7.1.3 ソフトウェアシステムと環境システム

ソフトウェアシステム（34 ページ「ソフトウェアシステム」参照）と環境システムは、プロジェクトコンフィギュレータの機能を用いてワークスペースの `Software/Software Systems` または `Environment/Environment Systems` フォルダ内に作成し、定義します。

ソフトウェアシステムと環境システムのインターフェースは、ファンクションのインターフェースと同じ方法で作成します（「ファンクション」参照）。またソフトウェアシステムと環境システムの標準レイアウトはファンクションのものと同

じです。ブロックのサイズはシグナルシンクとシグナルソースの数に応じて自動的に設定されますが、サイズと両側のシグナルシンクやシグナルソースの位置は任意に変更できます。

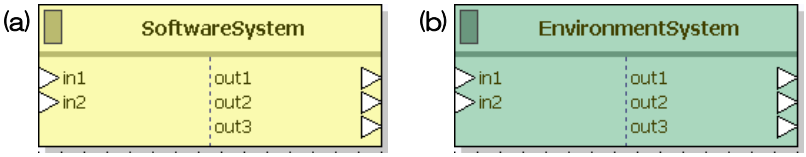


図 5-17 ソフトウェアシステム **(a)** と環境システム **(b)** の標準レイアウト

ソフトウェアシステムと環境システムには、任意の数のモジュール、SWC、ファンクションを割り当てて接続することができますが、ここでは以下のルールが適用されます。

- INTECRIO では、同じモジュール／ SWC の複数のインスタンスを使用することはできません。つまり、1 つのモジュール／ SWC は 1 つのソフトウェア／環境システム内において 1 箇所でしか使用できません。1 つのソフトウェア／環境システム内に同じモジュール／ SWC を含むファンクションが含まれている場合、また、すでにソフトウェア／環境システムに単体で組み込まれているモジュール／ SWC が、さらにファンクションの一部として組み込まれた場合も、コード生成でエラーメッセージが発行されます。
  ただし、グラフィックエディタ上では 1 つのソフトウェア／環境システム内に同じモジュール／ SWC を 2 個以上配置できるため、注意が必要です。
- 1 つのモジュール、SWC、ファンクションは、ワークスペース内においてソフトウェアシステムまたは環境システムのいずれか一方でのみ使用できます。たとえば `Software/Modules` フォルダ内にインポートされたモジュール／ SWC や `Software/Functions` フォルダ内に作成されたファンクションは、環境システムには組み込めません。

5.7.1.4 システムプロジェクト

ファンクションやソフトウェア／環境システムと同様に、システムプロジェクト（35 ページ「システムプロジェクト」参照）もプロジェクトコンフィギュレータの機能を用いてワークスペースの `Systems` フォルダ内に作成し、定義します。1 つのシステムプロジェクトには、ハードウェアシステム、ソフトウェアシステム、および環境システムをそれぞれ 1 つずつ割り当てます。

グラフィックエディタでは、ソフトウェアシステムと環境システム、および各ハードウェアデバイスがそれぞれ独立したブロックとして表示されます。未使用のシグナルはグラフィック表示から削除できます。

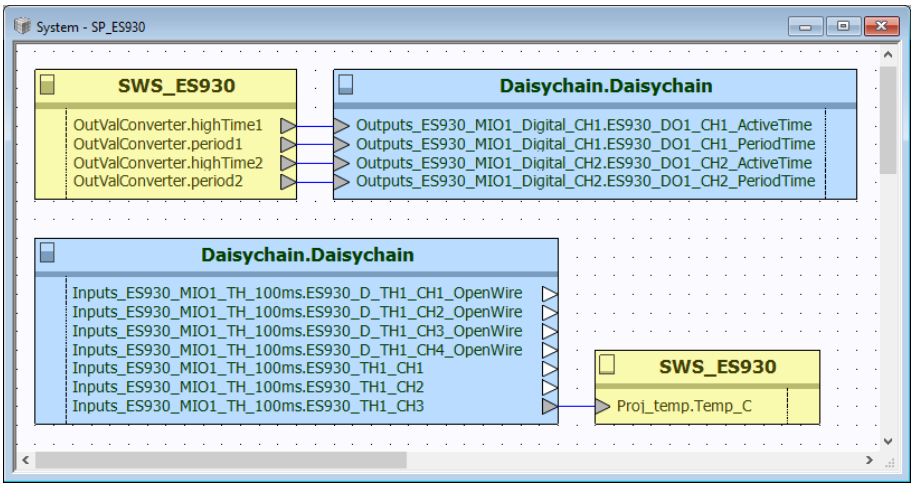


図 5-18 グラフィックエディタに表示されたシステムプロジェクト

ここで、ハードウェアとソフトウェアのシグナルソースとシグナルシンクを接続し、実験に必要なシグナルのルートを定義します。ここでも、ファンクション内の接続を行う場合と同じルール（「ファンクション」参照）が適用されます。

ハードウェアブロックのサイズとシグナルシンクやシグナルソースの位置は調整可能です。

ワークスペースには複数のシステムプロジェクトを定義できますが、実際に使用できるのは一度に 1 つのシステムプロジェクトのみです。ビルドオプションはシステムプロジェクトごとに個別に設定できるので、実行ファイルの作成条件を柔軟に変更することができます。

ビルド処理を実行するには、WS ブラウザでシステムプロジェクトを選択し、ショートカットメニューからビルドコマンドを選択します。メニューまたはツールバーからビルド処理を実行すると、現在アクティブになっているシステムプロジェクト（太字で表示されています）がビルドされます。

5.7.2 オンラインモード

オンラインモード（実験実行中）においては、動的コネクションを変更することができます。ただしモジュール、ファンクション、ソフトウェアシステムの内容や表示についての変更（シグナルソース／シンク、モジュール／ファンクションの削除や追加など）は行えません。

オンラインモードにおいて新たに接続されたシグナルソースとシグナルシンクの実装情報は、ターゲット上で実行される「コピー処理」内で調整されます。値の範囲や量子化が異なった場合、適切な変換が行われますが、シグナルシンクへの値の代入には常に制限条件があり、この制限を無効にすることはできません。

接続を編集する方法はオフラインモードの場合と同じです。さらに、オンラインモードではスティミュラスシグナルをシグナルシンクと接続することもできます。

5.8 OS コンフィギュレータ

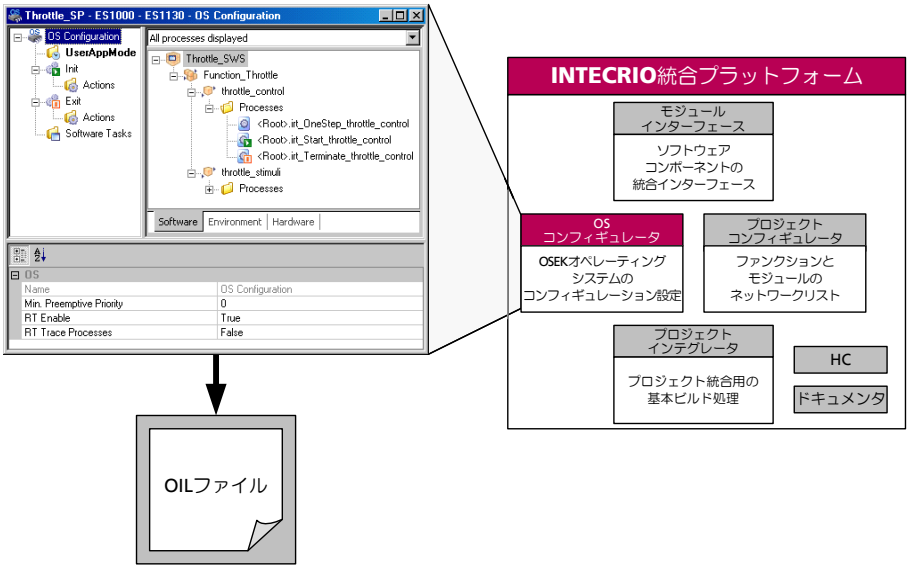


図 5-19 OS コンフィギュレータ

オペレーティングシステムのコンフィギュレーション設定は、リアルタイムプロトタイプの作成において非常に重要な作業です。INTECRIO では、この作業は「**OS コンフィギュレータ**」（統合プラットフォームに含まれるコンポーネントのひとつ）を用いて行います。

5.8.1 項ではオペレーティングシステムの役割について概説し、5.8.2 項では OS コンフィギュレータについて説明します。

5.8.1 オペレーティングシステムの役割

OSEK コンソーシアム[1] は自動車産業界で使用されるリアルタイムオペレーティングシステムの規格を策定し、その中に「OIL」という OSEK 実装言語が含まれています。OSEK 規格に準拠するオペレーティングシステムは、「OSEK オペレーティングシステム」と呼ばれます。

ES910 と ES830 では、OSEK と AUTOSAR[2] に互換のオペレーティングシステム RTA-OSEK が使用され、PC での仮想プロトタイピングにおいては RTA-OSEK for PC が使用されます。

OSEK オペレーティングシステムは、複数のプロセスの実行を管理します。CPU は一度に 1 つの命令しか実行できないため、CPU が 1 つしかないシステムにおいては複数のプロセスまたは RE（ランナブルエンティティ）を完全に同時に実行することはできません。そのため、オペレーティングシステムによる疑似並行処理（マルチタスキング）を行う必要があります。オペレーティングシステムは、プロセッサをめぐって競合するタスクやプロセス／ RE の実行順序を決定し、適切なタイミングでタスク の切り替えを行います。

1. 自動車エレクトロニクス向けオープンシステムおよびインターフェースの作業部会（ドイツ語で **O**ffene **S**ysteme und deren schnittstellen für die **E**lektronik im **K**raftfahrzeug）
2. **Aut**omotive **O**pen **S**ystem **Ar**chitecture の略。https://www.autosar.org/ を参照してください。

5.8.1.1 スケジューリング

「スケジューリング」は OSEK オペレーティングシステムの核となる機能です。スケジューラは、起動された複数のプロセス／ RE のうち、どれを最初に起動するかを決定します。この決定方法、いわゆる「スケジューリングアルゴリズム」は、システムのリアルタイム性と効率を左右する非常に重要な要素です。詳細なリアルタイム挙動と効率についての厳しい要求に対応するため、OSEK オペレーティングシステムは静的スケジューリングと動的スケジューリングを組み合わせ、さらに協調スケジューリングとプリエンプティブスケジューリングを組み合わせて使用します。

静的スケジューリング

「静的スケジューリング」については、スケジュールされるプロセス／ RE についてのすべての情報と制限事項がスジューリングアルゴリズムによって認識されます。ユーザー定義可能な制限事項としては、実行時間、デッドライン、起動周期、優先順位、相互排除があります。プロセス／ RE のすべての制限事項がシステム起動前にわかっているので、各プロセス／ RE の処理順序は前もってオフラインで決定されます。このような静的スケジューリングが定義されていれば、ランタイムには定義されている順序とタイミングでプロセス／ RE を起動するだけで十分です。

動的スケジューリング

一方、「動的スケジューリング」は、アルゴリズムで事前に認識されるのは「起動されるプロセス／ RE」の存在のみで、実際の起動順序や条件については何も認識されていません。ランタイムにおいて各プロセス／ RE は自然発生的に起動されるので、スケジューラはその都度どのプロセス／ RE を実行すればよいかを判断する必要があります。

動的スケジューリングは、外部イベントに柔軟に対応できるという点で静的スケジューリングより優れています。特に、短いレイテンシが要求される場合、静的スケジューリングの有効性は低くなります。一方、動的スケジューリングのデメリットは、より高度な計算能力が必要になることと、プロセス／ RE 管理のためにメモリ所要量が増えることです。OSEK オペレーティングシステムは、動的と静的の両方のスケジューリングをサポートしているので、レスポンスタイムとメモリ容量に関する要件に応じて、複数のスケジューリング方法を最適な形で組み合わせることができます。

5.8.1.2 タスク

「タスク」は静的スケジューリングの設定時に定義され、一連のプロセス／ RE が割り当てられます。各プロセスは、指定された順序で実行する必要があり、しかも所定の起動イベントが発生した時点で、定義された優先度に従って実行されなければなりません。

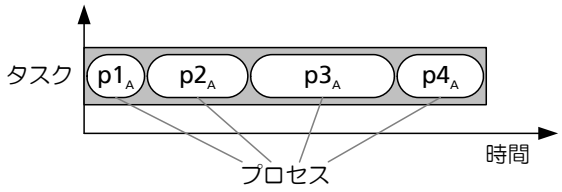


図 5-20 タスクのしくみ

ランタイムにおける動的タスクスケジューリング（「マルチタスキング」とも呼ばれます）は、タスク単位で行われます。タスク内のプロセス／ RE の処理順序は静的に定義されているので、スケジューラが決定する必要はありません。つまり、数の非常に多いプロセス／ RE ではなく、限られた数のタスクだけを管理すればよいので、ランタイムに必要な計算能力とメモリ容量は少なくて済みます。

各タスクには静的な「優先度」が割り当てられ、ランタイムにおいては、起動されたタスクはそれぞれの優先度に基づいて実行されます。つまり優先度の高いタスクが優先度の低いタスクよりも優先されます。複数のタスクに同じ優先度を持たせることもでき、同じ優先度のタスクが同時に起動されるようにスケジューリングされている場合は、それらのタスクは FIFO キューに登録され、「先着順」に処理されます。

動的スケジューリングは以下のようなタスクの状態モデルに基づいて実行されます。タスクは起動されると「アクティブ」（“activated”）状態になります。ここで、起動されたタスクの優先度が「実行中」（“running”）タスクの優先度よりも高く、かつタスクの切り替えが可能なタイミングであった場合は、起動されたタスクが実行され（このタスクの状態は「非アクティブ」（“inactive”）から直接「実行中」に遷移します）、カレントタスクの実行が中断されます（このタスクの状態は「アクティブ」に変わります）。「実行中」のタスクが終了すると、つまりステータスが「非アクティブ」に変わると、「アクティブ」タスクのうち優先度が最高で、FIFO キューの先頭位置に入っているタスクが実行開始（中断されていたタスクの場合は続行）されます。図 5-21 はタスクの状態と遷移を示しています。

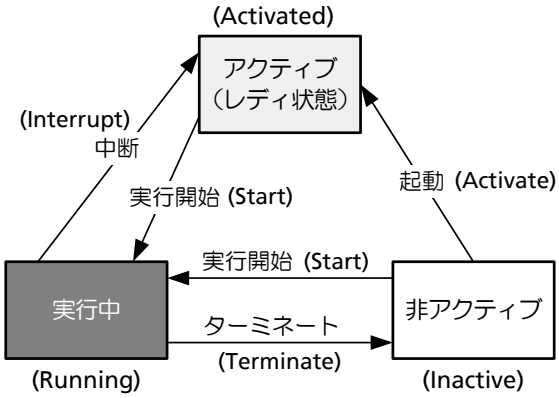


図 5-21 タスクの状態と遷移

5.8.1.3 協調スケジューリングとプリエンプティブスケジューリング

「実行中」タスクとそれより優先度の高い「アクティブ」タスクとを切り替える方法は 2 通りあります。1 つは、切り替えをソフトウェア内であらかじめ定義された位置（具体的にはプロセス／ RE の境界）で行う方法です。この場合、優先

度の高いタスクが、現在実行中の優先度の低いタスク内の現在のプロセス／ RE が終了するまで待つことになり、この方法は「協調スケジューリング」と呼ばれます（図 5-22 参照）。

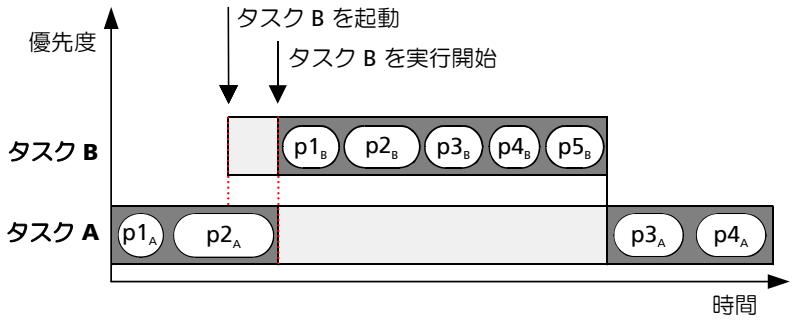


図 5-22 協調スケジューリング

注記

RTA-OSEK は協調スケジューリングをサポートしていません。

協調スケジューリングの利点は、リソースを効率的に利用できることです。スタック、レジスタ、メッセージなどのリソースへの排他的アクセスが保証されます。すべてのプロセス／ RE の実行に同じレジスタセットが使用されるので、タスク切り替え時に現在のプロセス／ RE のコンテキストを保存する必要がありません。一方、協調スケジューリングのデメリットは、プロセス／ RE の最大実行時間に応じてレスポンスタイムが比較的長くなってしまうことです。

第 2 の方法である「プリエンプティブスケジューリング」では、プロセス／ RE 内のマシン命令の境界でタスクを切り替えることができます（ただし、割り込みが実行されていない場合）。

そのため、スケジューラはタスク内の現在実行中のプロセス／ RE を中断して、そのタスクより優先度の高いタスクの実行を開始することができます（図 5-23 参照）。

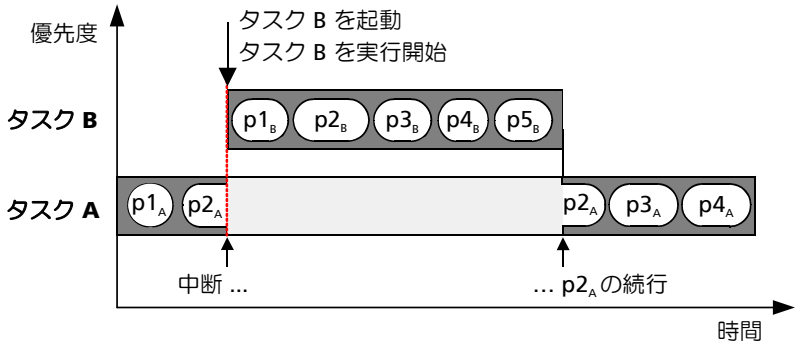


図 5-23 プリエンプティブスケジューリング

外部イベント（割り込み）や、被制御変数による周期的起動を実現するためには、レスポンスタイムを短くする必要がありますが、プリエンプティブスケジューリングであればこの要件を満たすことができます。ただしプリエンプティブスケジューリングの場合、中断されたプロセス／ RE のコンテキストを保護してデータの完全性を保証するためのメモリが必要となるため、メモリ所要量が多くなってしまうというデメリットがあります。

プリエンプティブスケジューリングでは、外部イベント、つまり割り込みを柔軟に処理できます。各割り込みソースに適切な優先度を割り当てることにより、非常に短いレスポンスタイムでタスクを起動することができます。このメカニズムにより、「非アクティブ」状態から「実行中」状態へ直接遷移することが可能となります（図 5-21 参照）。割り込み発生により呼び出され、割り込みコントローラにより制御されるタスクは、「ハードウェアタスク」と呼ばれます。

プリエンプティブタスクと協調タスクの優先度領域は互いに重なり合いません。プリエンプティブタスクの優先度は常に協調タスクの優先度よりも高くなっています。

5.8.1.4 プリエンプティブスケジューリングにおけるデータの完全性

プリエンプティブスケジューリングでは、優先度の低いプロセスまたは RE が優先度の高いプロセス／ RE により中断される可能性があります。図 5-24 のように、連続する 2 回の読み取り処理の間で割り込みが発生した場合、変数の 1 回目の読み取り後に、優先度の高いプロセス／ RE が同じ変数に書き込みを行ってしまう可能性があり、その場合はデータの不整合が生じます。次の図は、リソース保護が行われていない状態で実行中のプロセス／ RE が中断された場合に発生するデータの不整合を示しています。

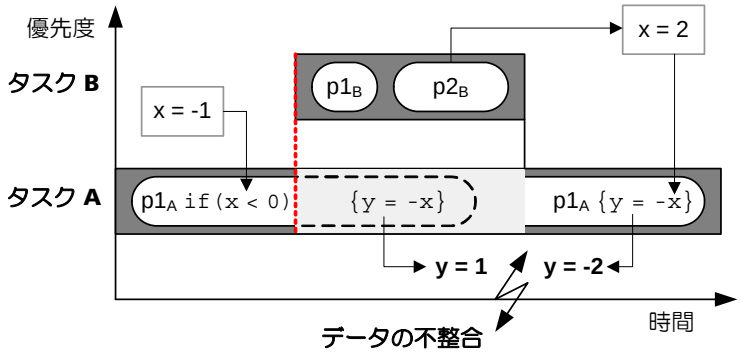


図 5-24 データの不整合

図 5-24 に示されるタスク A のプロセス $p1_A$ は、x の絶対値を計算します。

```
if (x<0)
  {y = -x;}
else
  {y = x;}
```

プロセス $p1_A$ はアルゴリズムの第 1 行を実行し、入力値（-1）を読み取ります。条件式の計算結果が真なので、y = -x という代入文の実行がスケジュールされます。しかしこの代入文が実行される前にプロセス $p1_A$ が中断され、タスク B のプロセス $p2_B$ が x に 2 という値を代入します。タスク B の終了後に $p1_A$ の実行が再開されると、$p1_A$ のアルゴリズムは、実行予定であった y = -x という代入文に x の値として -1 ではなく 2 を使用するため、$p1_A$ は間違った結果を出力します。つまり |-1| の結果として 2 が出力されてしまいます。

これはデータの不整合を説明するシンプルな例ですが、実際のアプリケーションにおいては、データの不整合によってシステムがクラッシュしてしまう危険性もあります。

システムの正当性はシステム内の割り込み処理の順序とタイミングによって決まります。上記の例のようなエラーをなくすためには、データの整合性を保証することが非常に重要です。

注記

プロセス P1（またはランナブルエンティティ R1）の開始から終了までの間、P1 ／ R1 がアクセスするメモリに格納されたすべてのデータは P1 ／ R1 によってのみ変更されるようにする必要があります。

メッセージ

上記の例のようなデータ整合性の問題を解決するため、「**メッセージ**」という概念がクロスバー（3.5.5 項参照）によってサポートされています。メッセージは「保護されたグローバル変数」で、グローバル変数のコピーを用いることによって「保護」が行われます。コピーが必要かどうかはシステムによって分析され、実行時間に大きな影響を与えない範囲での最適化されたデータ整合が行われます。

プロセスの開始時、入力メッセージはすべてそのプロセスのプライベート領域に「メッセージコピー」としてコピーされ、プロセス終了時には、これらのメッセージコピーがすべて出力メッセージ（グローバル領域）にコピーされます。この動作原理を図 5-25 に示します。

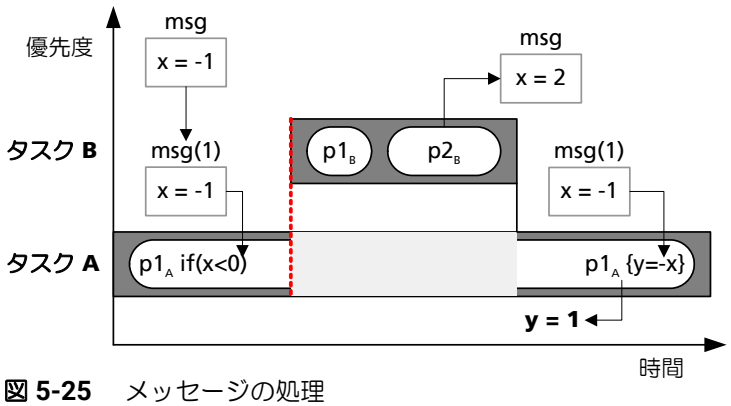


図 5-25 メッセージの処理

タスク A のプロセス $p1_A$ は、実行開始時に入力メッセージ `msg` をプライベートメッセージコピー `msg(1)` にコピーし、プロセス内で実行されるこのメッセージの読み取り処理は、すべてこのプライベートコピーを対象に行います。プロセス $p1_A$ はタスク B により中断され、`msg` の値はタスク B によって `-1` から `2` に変更されてしまいますが、この変更はプロセス $p1_A$ には影響しません。プロセス $p1_A$ は常にプライベートコピーを用いて以下のアルゴリズムを実行するため、データの整合性が保証されます。

```
if (x<0)
   {y = -x;}
else
   {y = x;}
```

AUTOSAR SWC を使用する場合は、クロスバーの代わりに AUTOSAR ランタイム環境（RTE、4.2 項参照）が使用されます。RTE は、OS のリソースや割り込みブロック機能などを利用し、転送データの整合性を保証します。

5.8.1.5 アプリケーションモード

「アプリケーションモード」は、システム全体の動作モードを実行時の状況に応じて変更できるようにするためのものです。これにより、完全に異なる機能を持つ複数のアプリケーションを柔軟に設計することができます。一般的なアプリケーションモードには、「始動モード」、「通常運転モード」、「シャットダウンモード」、「診断モード」、「フラッシュ書き込みモード」などがあります。個々のアプリケーションモードごとに、専用のタスク、優先度、タイマ設定などを定義できます。

> **注記**
>
> INTECRIO は RTA-OSEK を使用しているので、アプリケーションモードを 1 つしかサポートしていません。

アプリケーションモードは「初期化フェーズ」と通常の「シーケンスフェーズ」という連続する 2 つのフェーズで構成されます。初期化フェーズではすべての割り込みが無効になり、ここでハードウェアレジスタや変数の初期化などが行われます。そして初期化フェーズの最後に割り込みが起動され、各タスクの通常処理が開始されます。

5.8.2 OS コンフィギュレータの処理

INTECRIO の OS コンフィギュレータの処理の流れを図 5-26 に示します。

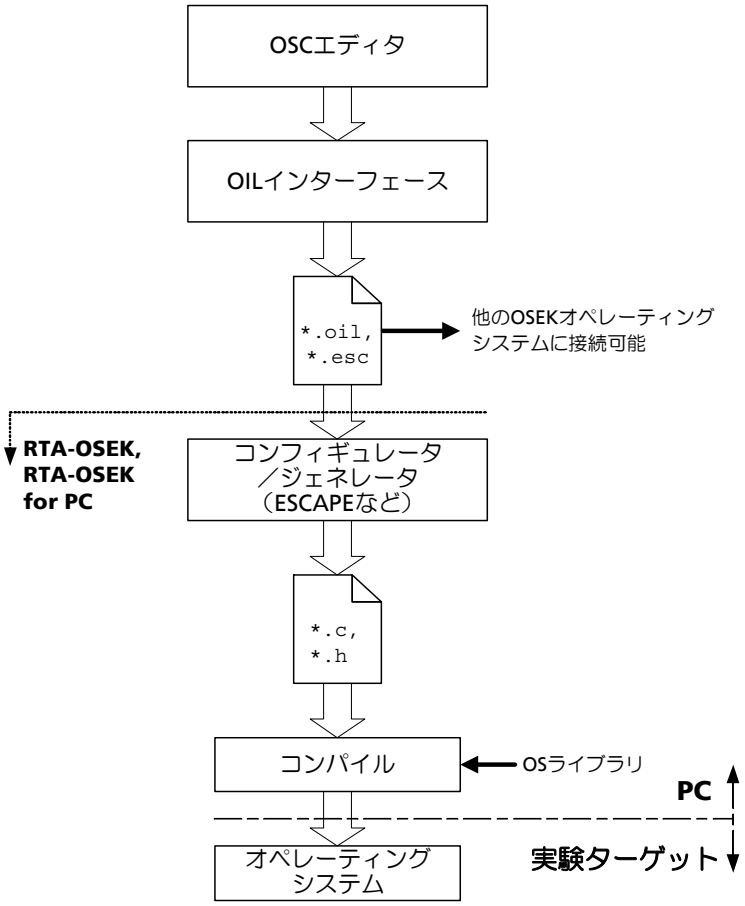


図 5-26 OS コンフィギュレータの処理の流れ

この図は、オペレーティングシステムのコンフィギュレーションを設定する際の処理の流れを表しています。

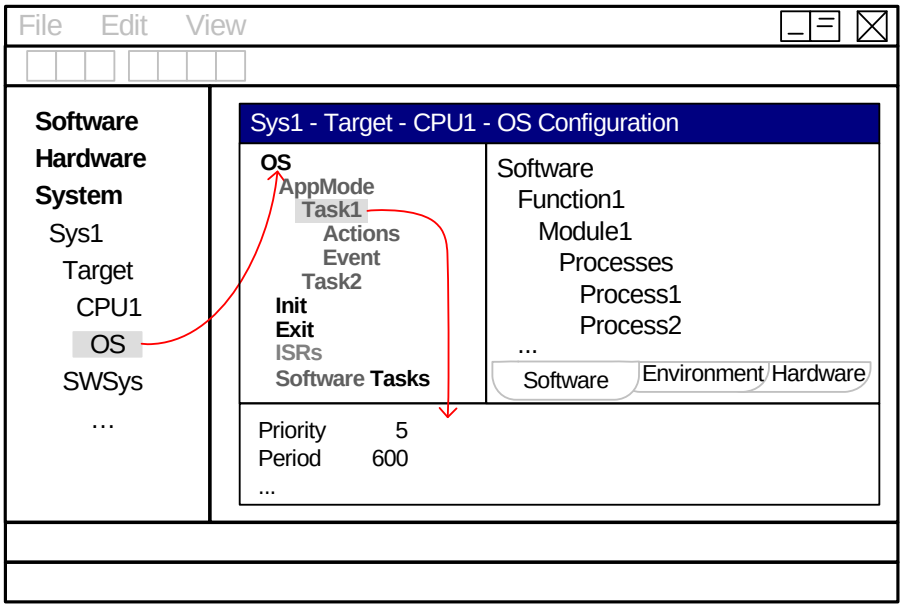
「OSC エディタ」は OS コンフィギュレータの専用エディタです。システムの内容がアプリケーション構成に従って表示されるので、システム全体を素早く把握でき、容易にコンフィギュレーションを編集することができます。

OSC エディタで設定された内容は、「OIL インターフェース」によって生成されるコンフィギュレーションファイル（`*.oil` ファイル）に、OIL 言語で記述されます。

なおここでは、OSEK コンフォーマンスクラスの BCC1 と BCC2 だけがサポートされています。OIL コンフィギュレーションファイルは、どの OSEK オペレーティングシステムにも対応しています。INTECRIO においては、ETAS のオペレーティングシステム（RTA-OSEK または RTA-OSEK for PC）が使用されます。

「コンフィギュレータ／ジェネレータ」によって `*.oil` ファイルから C コードファイルが生成されます。そしてこの C コードファイルがコンパイルされ、オペレーティングシステムライブラリとリンクされます。このようにして生成されたオペレーティングシステムの実行コードのみが実験ターゲットにロードされます。

5.8.3 OSC エディタ



OSC のユーザーインターフェースである「OS エディタ」は、3 つの部分に分かれています。左上のフィールドには OS コンフィギュレーションビューが表示され、右上にはソフトウェアシステム、環境システム、ハードウェアシステムの階層構造を示す 3 つのタブが表示されます。各タブにおいては、全プロセスを表示するか、またはタスクに割り当てられていないプロセスだけを表示するかを選択することができます。エディタの下部には、OS コンフィギュレーションビューで選択されているアイテムのプロパティを設定するフィールドがあります。

OSC エディタは以下の 2 種類のモードで使用されます。

- オフラインモード

  オペレーティングシステムのコンフィギュレーションはオフラインモードで設定します。オペレーティングシステムにはプリエンプティブタスクの最下位の優先度を設定できます。また、トレースツール RTA-TRACE を利用して実行状態をトレースするかどうかを指定することもできます。

  ASCET モデルの場合は、ASCET 上で設定したコンフィギュレーション（`*.oil` ファイル）を INTECRIO にインポートすることもできます。

- オンラインモード

  オンラインモード（実験実行時）における OSC エディタは、編集機能がブロックされ、表示ツールとしてのみ使用されます。

5.8.3.1 タスクの作成

OS コンフィギュレーションビューの最上位レベルに表示される “OS Configuration” がオペレーティングシステム全体を表します。

AUTOSAR SWC が含まれる場合

アプリケーションモードは作成されず、オペレーティングシステムの直下に初期化タスクと終了タスクが 1 つずつ作成されます。初期化タスクと終了タスクはすべてのアプリケーションモードによって使用されます。さらに必要に応じて `Software Tasks` フォルダにソフトウェアタスクを作成できます。タスクのタイプは、割り当てられたランナブルエンティティによって定義されます。

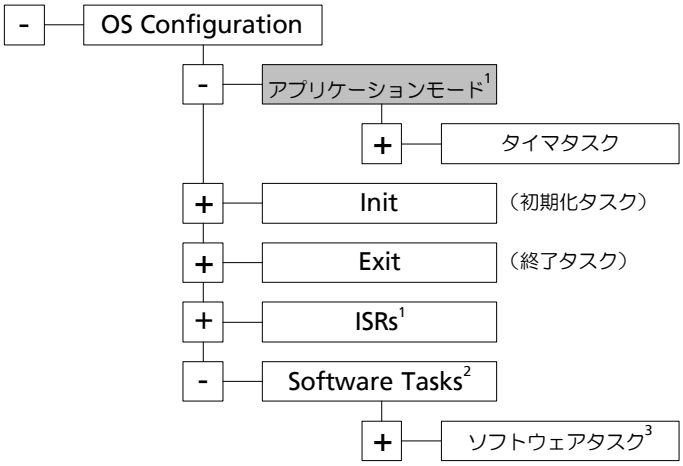


図 5-27 OSC：アイテムのツリー構造
1：AUTOSAR SWC の場合は存在しません。
2：AUTOSAR SWC の場合は `Tasks` になります。
3：AUTOSAR SWC の場合、タスクタイプは RE によって定義されます。

AUTOSAR SWC が含まれない場合

オペレーティングシステムの直下にアプリケーションモード（95 ページ参照）が作成されます。

アプリケーションモードのすぐ下のレベルにタイマタスクが作成され、オペレーティングシステムの直下に初期化タスクと終了タスクが 1 つずつ作成されます。初期化タスクと終了タスクはすべてのアプリケーションモードによって使用されます。さらに必要に応じて `Software Tasks` フォルダにソフトウェアタスク

を作成でき、 RTA-OSEK を使用している場合は、 ISRs フォルダに ISR（割り込みサービスルーチン）を作成することもできます。タスクタイプについては表 5-7 で説明します。

以下のタイプのタスクを使用できます。

タイマタスク （Timer タスク）	周期的に起動されるタスクで、タスクごとに異なる周期を設定できます。
ソフトウェアタスク （Software タスク）	オペレーティングシステムのコマンドまたは所定のイベントにより非周期的に起動されます。
初期化タスク （Init タスク）	自動的に作成されます。アプリケーションモードの開始時に 1 回だけ実行され、このタスクの実行時は割り込みが無効になります。
終了タスク （Exit タスク）	自動的に作成されます。シミュレーション終了時に 1 回だけ実行され、このタスクの実行時は割り込みが無効になります。
ISR	割り込みサービスルーチンです（RTA-OSEK のみ）。ハードウェアタスクとも呼ばれます。

表 5-7　タスクタイプ

注記

RTA-OSEK オペレーティングシステムは初期化タスクと終了タスクをサポートしていません。RTA-OSEK を使用する場合は、アプリケーションモードの開始時と終了時にこれらのタスクを C 関数として呼び出すコードが自動的に生成されます。

タイマタスクはアプリケーションモードの下位レベルに格納され、任意に作成、削除、名前の変更を行えます。各タイマタスクにはプロセスとイベントを割り当てるための Actions フォルダと Event フォルダが自動的に作成されます。

注記

仮想プロトタイピング（PC ターゲット）の場合、Event フォルダは作成されません。

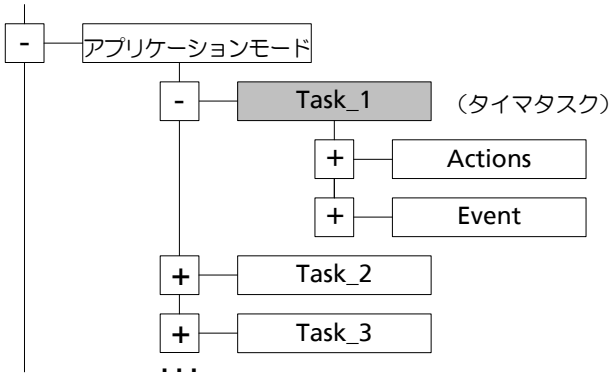


図 5-28　OSC：アプリケーションモードとタイマタスク

ソフトウェアタスクは Software Tasks フォルダ内に作成します。ソフトウェアタスクはどのアプリケーションモードにも属さず、オペレーティングシステムのスケジューリングにも含まれませんが、ハードウェアコンフィギュレータのイベントなどにより呼び出すことができます。

ISR（割り込みサービスルーチン）は ISRs フォルダ内に作成します。ISR の並び順はランタイムの挙動に影響しません。

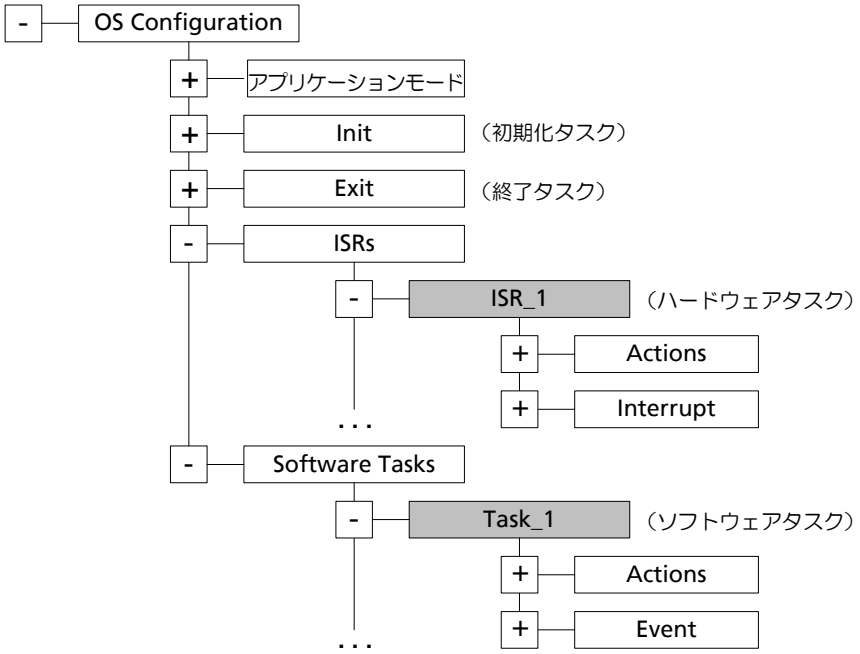


図 5-29 OSC：ISR とソフトウェアタスク（仮想プロトタイピングの場合、Event フォルダと ISR フォルダは存在しません）

5.8.3.2 タスクのプロパティ

タイマタスクとソフトウェアタスクはプロパティを編集することができます。不適切な値を設定しようとすると、警告やエラーメッセージが発行され、既存の値が保持されます。

> **注記**
>
> 初期化タスクと終了タスクのプロパティは編集できません。ISR については優先度のみ編集できます。

タスクのプロパティには以下のオプションがあり、AUTOSAR SWC の有無や、OS とターゲットのタイプにより、実際に使用されるオプションが異なります。

レガシー AUTOSAR SWC が含まれる場合

- Priority（優先度）

  複数のタスクが起動されている場合、各タスクの優先度により次に実行されるタスクが決定されます。現在実行中のタスクは、それよりも優先度の高いタスクが起動されると、中断されます。

  優先度として設定できる値の範囲は、OS のタイプとタスクのスケジューリングタイプにより異なります。100 ページの図 5-30 に、設定できる値の体系を示します。

- Task Activation（同時に起動できる最大数）

  タスクをどの程度頻繁に起動できるかを決定するための値です。タスクの起動後、実行が終了する前に再び同じタスクが起動されると、そのタスクは 2 回起動されたことになります。システムリソースの節約のため、一度に起動できる数をここで制限することができます。

AUTOSAR SWC が含まれない場合

- TaskID（タスク ID）

  タスクを識別するためのユニークな番号

- Priority（優先度）

  複数のタスクが起動されている場合、各タスクの優先度により次に実行されるタスクが決定されます。たとえば、複数の周期タスクが同時に起動された場合、優先度が一番高いタスクが最初に実行されます。現在実行中のタスクは、それよりも優先度の高いタスクが起動されると、中断されます。

  優先度として設定できる値の範囲は、OS のタイプと、タスクのスケジューリングタイプにより異なります。図 5-30 に、設定できる値の体系を示します。

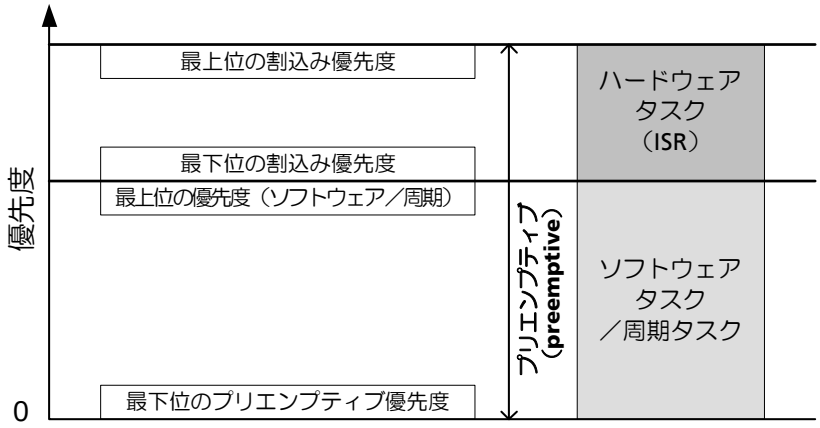


図 5-30 優先度の構造

> **注記**
>
> RTA-OSEK はプリエンプティブスケジューリングのみサポートしています。

- Period（周期）

  タスクが起動される周期を定義します。

  このオプションはタイマタスクについてのみ設定できます。

- Delay（遅延）

  タスクの起動を、何秒後に開始するかを指定します。この値が 0 の場合、このタスクはプログラムの起動直後に起動され、その後は各周期ごとに起動されます。

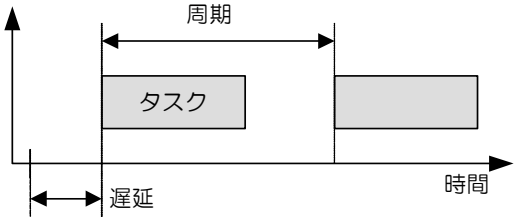


図 5-31 タスクの遅延

  このオプションはタイマタスクについてのみ設定できます。

- Execution Budget（実行バジェット）

  タスクの最大許容実行時間（秒数）を設定します。設定できる値の範囲は 0.000001 ～ 128 です。値を 0 にすると実行時間の監視が無効になります。

- Max. Number of Activation（同時に起動できる最大数）

  タスクをどの程度頻繁に起動できるかを決定するための値です。タスクの起動後、実行が終了する前に再び同じタスクが起動されると、そのタスクは 2 回起動されたことになります。システムリソースの節約のため、一度に起動できる数をここで制限することができます。

- Monitoring（監視）

  このオプションで、タスクの監視情報を収集する（`True`）かしない（`False`）かを指定します。このオプションが有効になっていると、監視用の変数（たとえばタスクの総実行時間を示すものなど）が追加されます。これらのシグナルソースの情報は ASAM-MCD-2MC ディスクリプションにも出力され、ETAS Experiment Environment または INCA ／ INCA-EIP で測定することができます。

  監視を有効にすると、タスク専用の以下の監視変数が生成されます。

変数	値の意味
`actTime`	タスクの起動時刻
`startTime`	タスクの開始時刻
`grossRunTime`	タスクの合計実行時間
`netRunTime`	タスクの正味の合計実行時間
`minRunTime`	タスクの最短実行時間
`maxRunTime`	タスクの最長実行時間
`dT`	タスクの前回の起動と今回の起動の時間間隔

  監視変数はシステムチックを単位として測定されるので、これを秒単位に変換して使用します。

- Exclude from Tracing（トレースから除外）

  タスクを RTA-TRACE による監視対象から除外する（`True`） かしない（`False`） かを指定します。

5.8.3.3 タイマタスクとソフトウェアタスクのセットアップ

ソフトウェア／環境システムに含まれるモジュールや SWC、ファンクションの中で使用されるプロセスと RE（AUITOSAR ランナブルエンティティ）は、タスクの Actions フォルダに挿入します。またハードウェアコンフィギュレーションに基づいて自動生成されたプロセス／ RE（ハードウェア初期化やシグナル処理など）もこのフォルダに割り当てることができます。1 つのプロセス／ RTE を複数のタスクに割り当てることはできません。

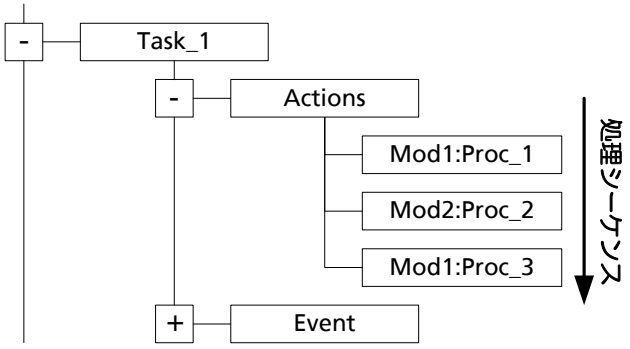


図 5-32 OSC：プロセスが割り当てられたタスク

> **注記**
>
> RTA-OSEK および RTA-OSEK for PC オペレーティングシステムは、「プロセス」をサポートしていないため、割り当てられたプロセスを C 関数として呼び出すコードが INTECRIO によって自動生成されます。

タスクに割り当てられたプロセスと RE は、OSC エディタ上に表示されている順で処理されます。つまり一番上に割り当てられたプロセス／ RE が最初に処理され、一番下のプロセス／ RE が最後に処理されます。OSC エディタ上でプロセスの位置を変えることによって実行順を変更できます。

1 つのタスクの Event フォルダには 1 つのイベントだけを割り当てることができます（RTA-OSEK for PC/PC の場合を除きます）。「イベント」は割り込みにより発生するトリガアクションで、Event フォルダに割り当てられたイベントの発生によって当該タスクが 1 回だけ起動されます。イベントには、ハードウェアドリブンのイベント（ウォッチドッグアラームなど）や非周期的イベント（トリガシグナル受信時に発行されるものなど）があります。

> **注記**
>
> Event フォルダには割り込みによりトリガされるシグナルだけを 1 つだけ割り当てることができます。

プロセスの割り当ては、オートマッピング機能により自動的に行うことができ、マニュアル操作によるマッピングも可能です。

5.8.3.4 ISR（割り込みサービスルーチン）のセットアップ

ISR のセットアップの方法はタスクの場合と似ています。ハードウェア割り込みは ISR の Interrupt フォルダに割り当てられます。1 つの HW 割り込みは 1 つの ISR にのみ割り当てることができます。ISR に属するプロセスは Actions フォルダに挿入します。ISR の場合は、タスクの場合と異なり、Actions フォルダ内に割り当てられたプロセスの位置は意味を持ちません。

一部の HW 割り込みには **AnalyzeCapable** プロパティ（ハードウェアコンフィギュレータにより自動設定されるプロパティ）があります。このような場合、トリガイベント（割り込み）にサブイベントを持たせて任意のプロセスを割り当てることにより、ランタイムに分析を行うことができます。

このような HW 割り込みが ISR に割り当てられると、`Actions` フォルダ内に `Event Dependencies` サブフォルダが自動的に作成されます。 `Event Dependencies` フォルダにはサブイベントが含まれ、各サブイベントは、プロセスを格納するサブフォルダとして表示されます。

これらのアイテム構成は以下のようになります。ランタイムにおいて ISR が実行され、その際、分析機能によって `Event Dependencies` フォルダ内にイベントが割り当てられていることが認識されると、それらのイベントのプロセスリストが実行されます。

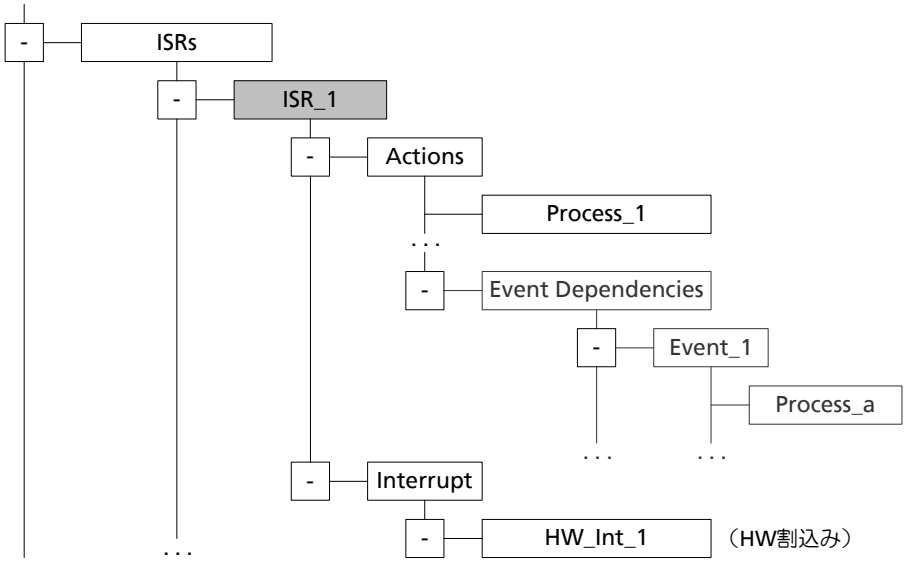


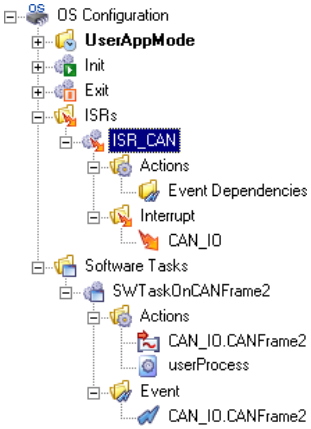
図 5-33 OSC：ISR のコンフィギュレーション

ISR のコンフィギュレーションには 2 とおりの構成方法がありますが、オートマッピングを行うと、デフォルトの構成が採用されます。

A デフォルト構成

特に厳密なスケジューリング要件に対応しなければならない場合以外は、この方法を使用してください。

HW 割り込みは新規に作成された ISR に割り当てられます。優先度の高い ISR 内で割り込みの分析処理だけが実行され、必要に応じて分析タスクがソフトウェアタスクを起動します。これらのソフトウェアタスク（ISR より優先度が低いタスク）に実際のプロセス（シグナルグループとユーザー定義プロセス）が割り当てられます。

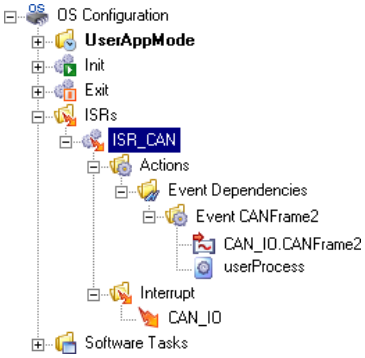


B. 最短レスポンスタイム用の構成

このコンフィギュレーションは、HW 割り込みをできるだけ短い時間で評価しなければならない場合に限り使用してください。

ISR を作成して HW 割り込みを割り当て、その割り込みに対応するイベントを `Event Dependencies` フォルダに割り当てます。 そしてシグナルグループとユーザー定義プロセスを `Event <event name>` サブフォルダに挿入します。

この構成のデメリットは、ISR に割り当てられた高い優先度でユーザー定義プロセスが実行されるため、その間、システム内の他の処理がブロックされてしまう、という点です。



5.9 プロジェクトインテグレータ

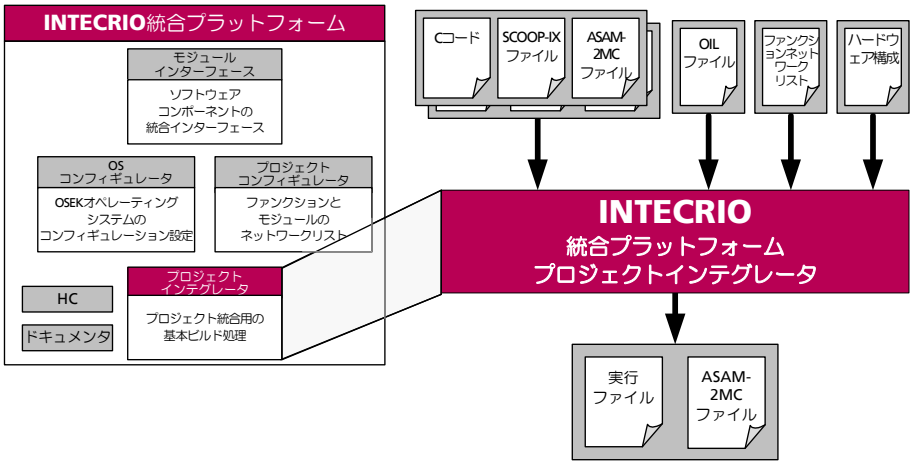


図 5-34 プロジェクトインテグレータ

プロジェクトインテグレータ（PI）は、システムのすべてのコンポーネント（モジュールとファンクション、ハードウェアインターフェース、OS コンフィギュレーションなど）を 1 つの実行ファイルに一体化します。プロジェクトインテグレータは複数のコンポーネントで構成された「ツールチェーン」のようなもので、Make 機能を実現する PI ビルドシステムがその中核となっています。

その他のコンポーネントには、ビルド設定を行う PI プラグイン、コンパイラ、リンカなどがあります。

ビルド処理を実行すると、実行ファイル以外に ASAM-MCD-2MC ファイルも生成されます。このファイルは、各モジュールの ASAM-MCD-2MC ファイルと、ラピッドプロトタイピングプロジェクト用の情報を一体化したものです。また、オプション設定により、RTA-TRACE 用のコンフィギュレーションファイルも生成できます。

プロジェクトコンフィギュレータとは異なり、プロジェクトインテグレータには専用のエディタはありません。グラフィカルフレームワークにおいては、**Integration** メニューからプロジェクトインテグレータのコマンドを実行できます。

5.9.1 ビルド処理

正常なビルド処理を行うためには、すべてのモジュールについて所定のファイル（`*.six`、 `*.c`、 `*.h`、 `*.a2l` など）が用意されている必要があります。これらのファイルがあれば、ファイル生成に使用した BMT は必要はありません。

5.9.1.1 概要

図 5-35 は、ビルド処理の流れと、各フェーズで使用されるファイルのタイプを図解したものです。各フェーズは、さらにいくつかのステップに細分されます。

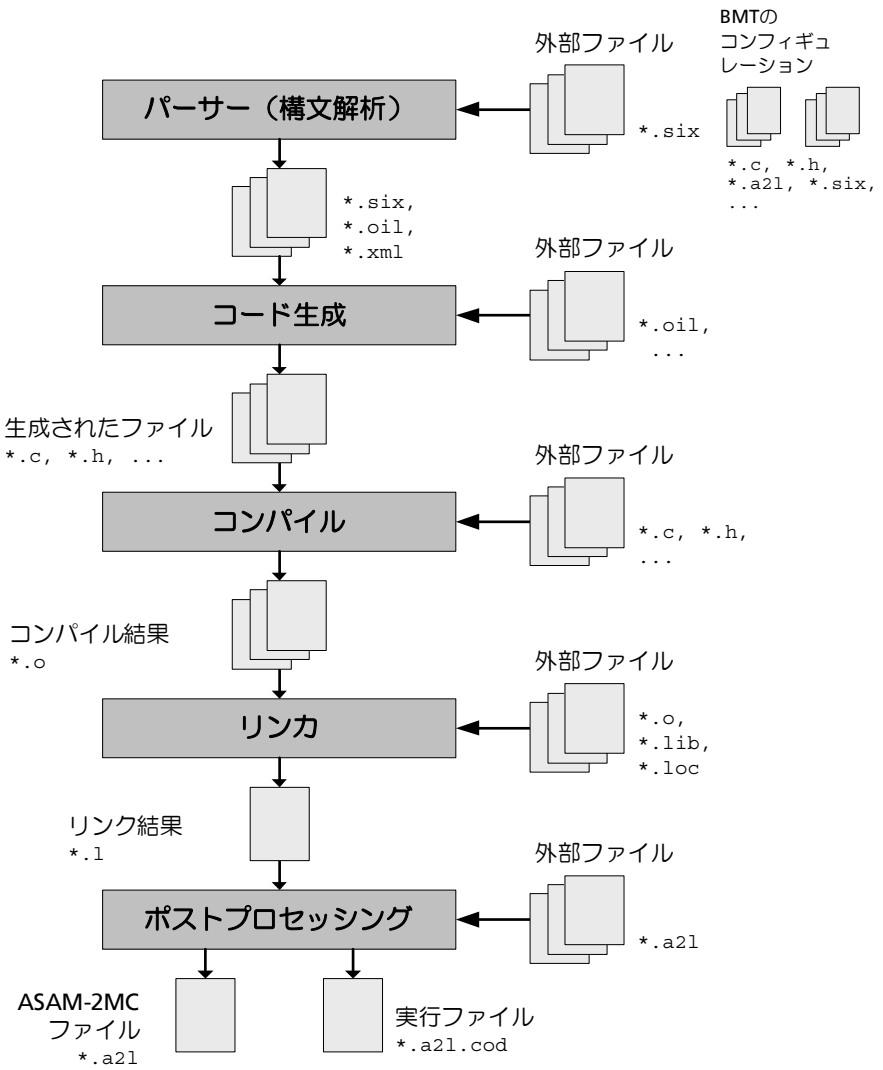


図 5-35 ビルド処理

ビルド処理を開始すると、「パーサー」がすべてのコンポーネントのコンフィギュレーションとプロジェクト全体のコンフィギュレーション、さらに BMT から提供されたインターフェースディスクリプション（*.six）を分析します。

「コード生成」では、インターフェースディスクリプション（*.six）と、各 INTECRIO コンポーネントごとに生成されたコンフィギュレーション（*.oil）をもとに C ソースファイルが生成され、モジュールの機能コードとその他の情報を含む C コードが出力されます。

「コンパイル」では、コード生成で出力されたソースファイルと BMT から提供されたソースファイルから、オブジェクトファイルが生成されます。ここで外部 C コードファイルやヘッダファイルを追加することができます。1 つのシステムプロジェクトに対しては 1 つのコンパイラしか使用できません。

コンパイル結果は「リンカ」により 1 つのバイナリファイルにリンクされます。ここでは外部オブジェクトファイルやライブラリを追加することができます。1 つのシステムプロジェクトには 1 つのリンカしか使用できません。

「ポストプロセッシング」では、バイナリファイルから、プロジェクト全体の実行ファイルと ASAM-MCD-2MC ディスクリプションファイルが作成されます。BMT で生成された ASAM-MCD-2MC ファイルはここで追加されます。

5.9.1.2 処理の詳細

ビルド処理を実行する際は、システムプロジェクトに以下のコンポーネントが正しく定義されていることを確認する必要があります。

- ハードウェアのコンフィギュレーション（5.3 項参照）
- ソフトウェアのコンフィギュレーション（5.7 項参照）
- システムプロジェクト（5.7 項内の「システムプロジェクト」参照）
- OS コンフィギュレーション（5.8 項参照）

ビルド処理を開始すると自動的に処理が実行され、処理のフェーズごとに状態や警告／エラーメッセージが表示されます。警告が発生しても処理が続行されますが、一般的なエラーが発生した場合は、そのフェーズが完了した時点で処理が中止されます。重大なエラーが発生した場合やソースファイルが見つからない場合は、ビルド処理はフェーズの途中であっても直ちに中止されます。

生成された実行ファイルはターゲットの RAM 上で実行でき、ランタイムにおいて動的に再構成することができます。また、オプション設定により FLASH メモリ上で実行されるコードを生成することもできます。FLASH コードは動的に再構成することはできませんが、これを用いてラピッドプロトタイピングハードウェアをスタンドアロンモードで稼動させることができます。コード生成フェーズは、単独で実行することも可能です。

ビルド処理中で作成されたファイルは、各システムプロジェクト専用のディレクトリ（`<workspace>¥cgen¥system<n>`）に書き込まれます。このディレクトリは、コンパイル後のファイルやその他一時ファイルなどに使用される複数のサブディレクトリで構成されます。最終的な実行ファイルと ASAM-MCD-2MC ディスクリプションファイルは、デフォルト状態においては、プロジェクトディレクトリ（`...¥<workspace>`）の `Results` サブディレクトリに書き込まれます。実行ファイル（`<basic name>.a2l.cod`）と ASAM-MCD-2MC ファイル（`<basic name>.a2l`）のベースネームは、デフォルトではシステムプロジェクト名が使用されますが、必要に応じて変更もできます。

デフォルト状態においては、ビルド処理は差分についてのみ実行されます。つまり入力ファイルの変更内容がチェックされ、不足している出力ファイルや、変更されている入力ファイルから生成された出力ファイルだけが生成されます。必要に応じて、ファイルのクリーンアップ（BMT で生成されたコードと SCOOP-IX ファイルを除くすべての INTECRIO システムファイルを削除します）を行い、プロジェクト全体を完全にビルドすることもできます。また、実際のビルド処理を行わずにファイルのクリーンアップだけを行うことも可能です。

5.9.2 ASAM-MCD-2MC の生成

プロジェクトインテグレータのビルド処理では、ASAM-MCD-2MC のバージョン 1.4[1] に準拠する ASAM-MCD-2MC ディスクリプションファイル（`*.a2l` ファイル）が生成されます。測定／適合の対象となるモデルエレメントについて、ETAS Experiment Environment において必要な情報がこのファイルに記述されます。

1. 最新の仕様についてはこちらをご覧ください：https://www.asam.net/

この ASAM-MCD-2MC ファイルの生成時には、プロジェクトに含まれるモジュール用に BMT で生成された *.a2l ファイルが結合され、さらにプロジェクト全体についての記述も追加されます。

一般に、ASAM-MCD-2MC ファイルには以下の情報についてのディスクリプション（記述）が含まれています。

- プロジェクト情報
- プロジェクトで使用されるデータ構造体
- 変数（測定変数）とパラメータ（適合変数）
- 外部インターフェース
- 通信プロトコル
- 変換式

5.10 ETAS Experiment Environment（ETAS 実験環境）

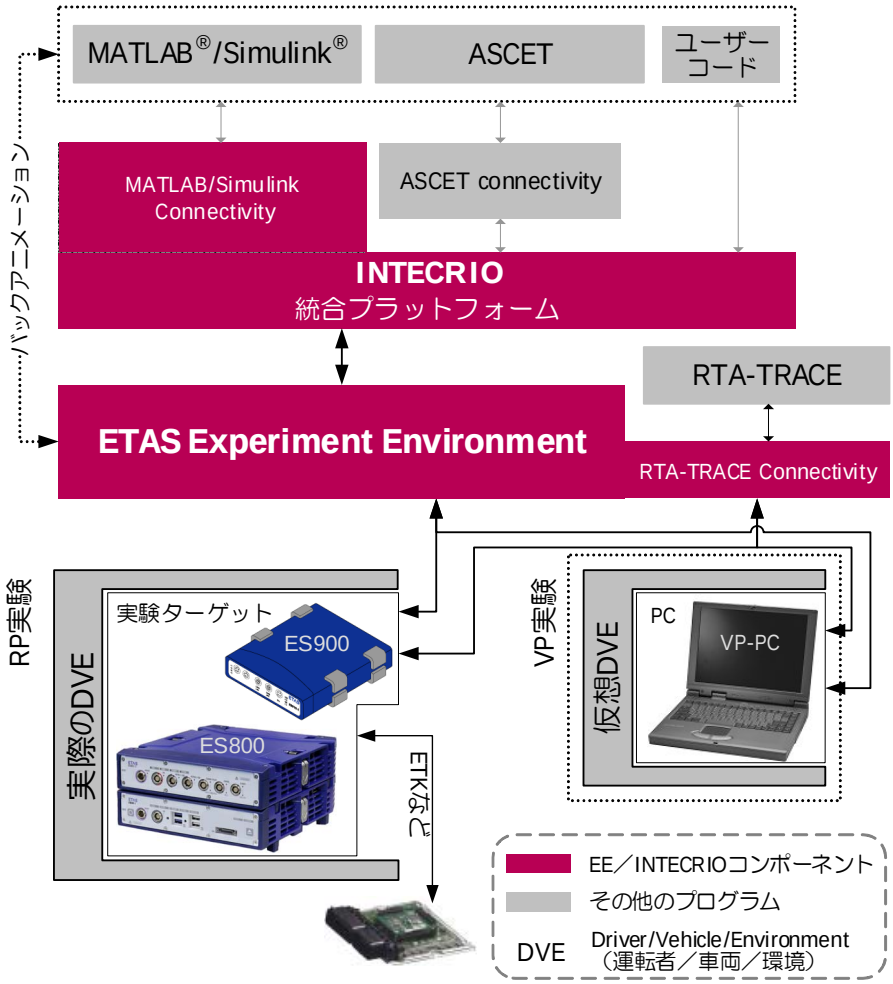


図 5-36 INTECRIO と ETAS Experiment Environment

本項では、ラピッドプロトタイピング実験の全般、および ETAS Experiment Environment について概説します。ETAS Experiment Environment の詳しい操作方法などは、オンラインヘルプを参照してください。

ETAS Experiment Environment で行う実験は、以下の 3 ステップに大きく分けられます。

- 実験の準備
- 実験の実行
- 実験後の分析

「実験の準備」のステップでは、プロトタイプの作成や、実験の環境設定などを行い、「実験後の分析」では、分析ツール[1] を用いた実験データの分析を行います。ここでは「実験の実行」について説明します。

ターゲットとして VP-PC を使用するプロトタイプではメモリページは 1 ページのみですが、ES910 または ES830 の場合は、2 ページ（リファレンスページとワーキングページ）を使用することができます。ただし ETAS Experiment Environment は複数のメモリページをサポートしていないため、2 ページを使用するには INCA/INCA-EIP を実験環境として使用する必要があります。メモリページついての詳細は、INCA と INCA-EIP のドキュメントを参照してください。

注記

ES800 ハードウェアシステムを用いた実験を行うには、INCA V7.2.17 以降の INCA と INCA-EIP、または V3.7.7 以降の ETAS 実験環境が必要です。

5.10.1 検証と評価

ファンクション開発の工程において、ソフトウェアコンポーネントやアプリケーションソフトウェア全体についての検証（テスト）や評価（妥当性確認）を行うには（3.4 項「開発プロセスにおける INTECRIO の役割」参照）、専用の実験環境が必要となります。

一般的に、以下のような作業を行える実験環境が必要です。

- コードとデータをターゲットにロードする
- 実験を開始／終了／一時停止する
- 各種エレメントをさまざまな手段で測定／適合する
- 必要に応じてスティミュラス（シミュレーション用データ入力）を使用する

ただし実際に行う作業は、実験を行うターゲットにより異なります。

また、効果的な実験を行えるよう、プロトタイプ作成時にさまざまなコンフィギュレーション設定を変更できることも重要です。

5.10.2 測定と適合

一般に、実験において行う主な作業は「測定」と「適合」です。ここで「測定」とは、エレメントの現在の状態を読み取り、さまざまな形式で視覚化することを意味し、「適合」とは、エレメントの状態を変更してシステムの挙動を調整することを意味します。ETAS Experiment Environment には、この「エレメント状態の測定／視覚化」と「エレメント状態の適合（調整）」という 2 つの機能が備わっています。

1. ETAS の測定データアナライザ（MDA）など

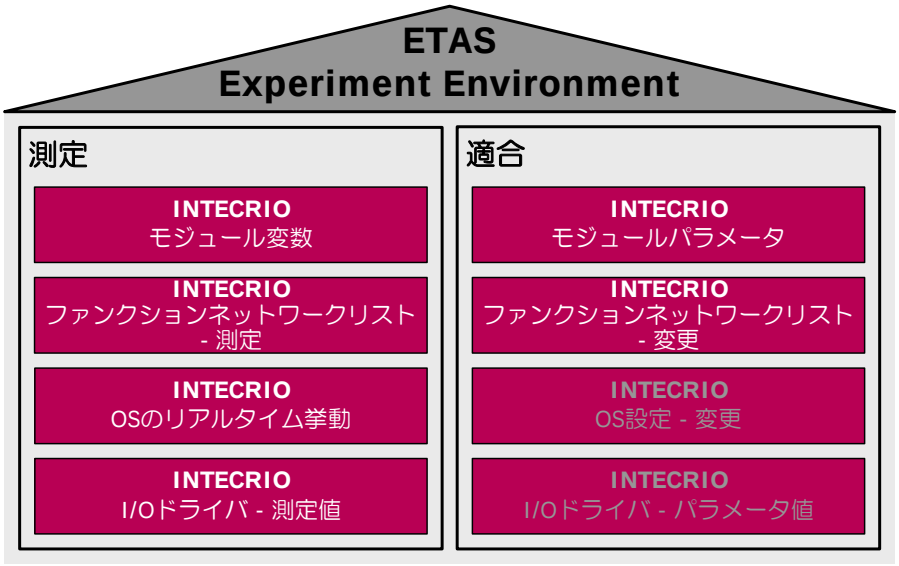


図 5-37 ETAS Experiment Environment の主要な機能

ETAS Experiment Environment では、以下のようなエレメントの測定が可能です。

- **モジュール変数** – BMT で記述されたモジュールに含まれる変数
- **ファンクションネットワークリスト** – ソフトウェア／ハードウェアモジュール間の接続で受け渡される値を測定することにより、コンポーネント間で交換されるシグナルの現在の情報を得ることができます。
- **オペレーティングシステムのリアルタイム挙動** – オペレーティングシステムの現在の状態（カレントタスク、実行時間など）
- **I/O ドライバ** – ドライバに受け渡しされる値や、ハードウェアのコンフィギュレーション

また、以下のようなエレメントを適合できます。

- **モジュールパラメータ** – 各モジュールのパラメータ
- **ファンクションネットワークリスト** – ソフトウェア／ハードウェアモジュール間の接続のルーティングをランタイムにおいて変更できます。これは検証や検証を行うには不可欠な機能です。

適合後のパラメータは、保存して再ロードすることができます。

5.10.3 各種ターゲットでの実験

ターゲット上で実験を実行するには、実際に行う作業（5.10.1 項「検証と評価」「妥当性確認と検証」参照）に対応するさまざまなインターフェースがターゲット上に用意されている必要があります。

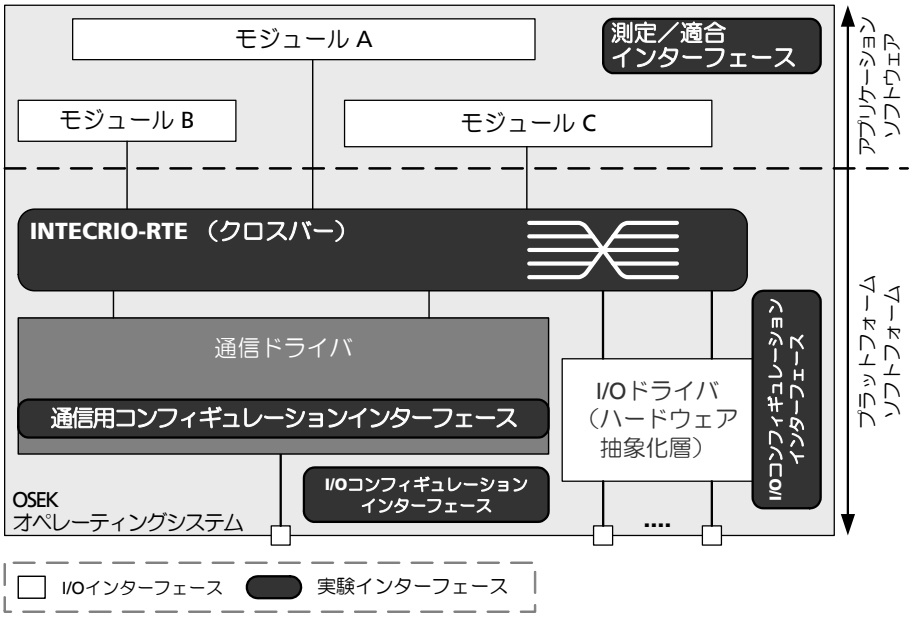


図 5-38 実験のインターフェース

図 5-38 は、図 5-37 に示した作業を行うために必要な各種実験インターフェースを示しています。

- **測定／適合インターフェース** – 変数やパラメータにアクセスするためのインターフェース
- **INTECRIO-RTE（クロスバー）** – ファンクションネットワークリストにアクセスするためのインターフェース
- **通信用コンフィギュレーションインターフェース** – I/O ドライバの値にアクセスするためのインターフェース
- **I/O コンフィギュレーションインターフェース** – I/O ドライバのコンフィギュレーションにアクセスするためのインターフェース
- **OS コンフィギュレーションインターフェース** – OS コンフィギュレーションにアクセスするためのインターフェース

さまざまな条件設定が必要なラピッドプロトタイピング実験を行うリアルタイムターゲットについては、上記のすべてのインターフェースをサポートしている必要があります。

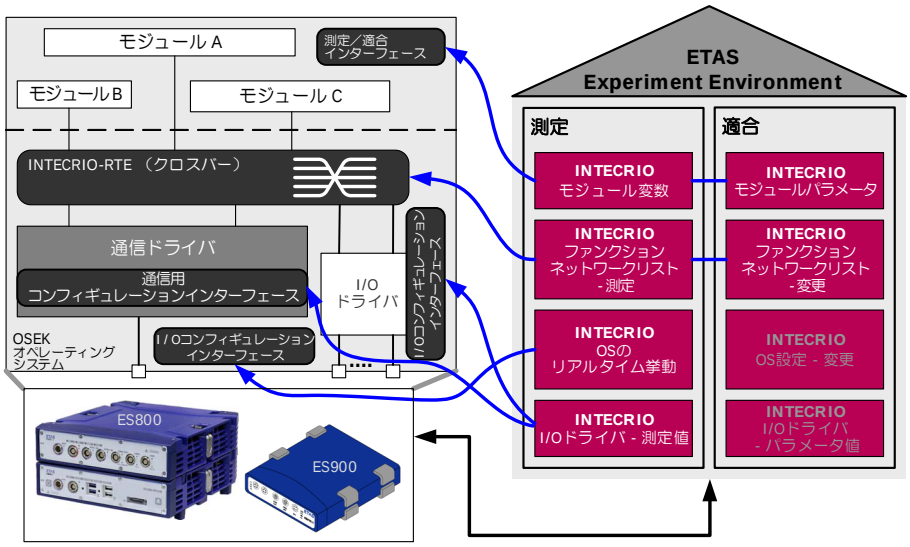


図 5-39　ラピッドプロトタイピングターゲット用実験インターフェース

図 5-39 は、各インターフェースがサポートするラピッドプロトタイピングシステムの測定／適合作業を示しています。

しかし量産用 ECU をターゲットとする実験では、オペレーティングシステムのリアルタイム挙動以外は、コンフィギュレーションを適合したりすることはありません。そのため、必要なのは測定／適合インターフェースと OS コンフィギュレーションインターフェースだけで、それ以外のインターフェースは必要ありません（図 5-40 参照）。

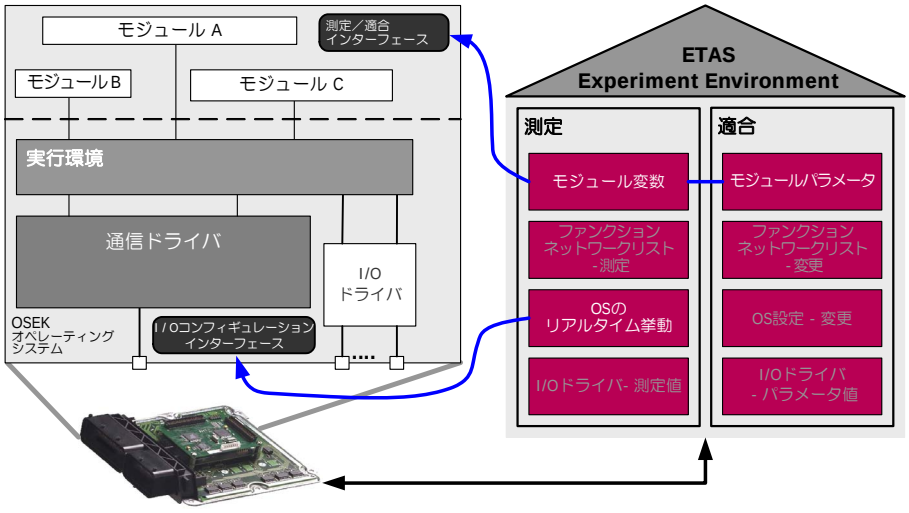


図 5-40　量産用 ECU の実験インターフェース

5.10.4 ETAS Experiment Environment を用いたラピッドプロトタイピング実験

基本的に、ラピッドプロトタイピングシステムを用いる実験には「バイパス実験」と「フルパス実験」の 2 種類があります。実際の操作はどちらの場合もほぼ同じで、プロトタイプの実行、値の測定、パラメータの適合などの操作を行いますが、両者の間には以下のような明確な相違点があります。

5.10.4.1 バイパス実験

「バイパス実験」は、アプリケーションソフトウェアの一部分をラピッドプロトタイピングハードウェア上で実行する実験です。

これは、すでにソフトウェアファンクション全体の検証が終了している ECU について、その一部のソフトウェアファンクションのみを開発するような場合に適しています。バイパス実験を行うには、ECU の既存のソフトウェアを、バイパスインターフェースをサポートするように変更する必要がありますが、この変更は、「バイパスフック」を追加するだけなので、比較的容易に行えます。

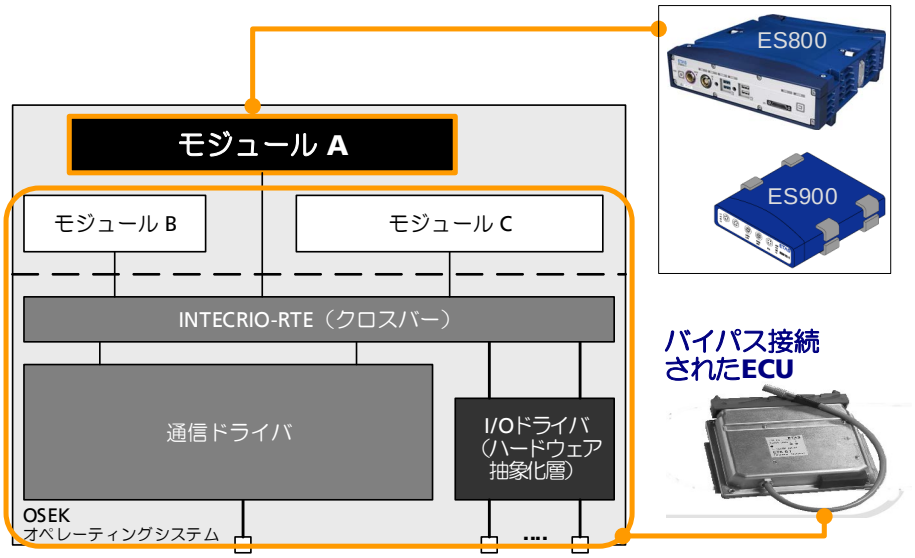


図 5-41 バイパス実験のしくみ

バイパスシステムは、「互いにリンクしあう 2 つのプロセッサを持つシステム」として考えることができます。プロセッサの 1 つはバイパス接続された ECU で、もう 1 つはラピッドプロトタイピングシステムです。アプリケーションソフトウェアや、場合によってはプラットフォームソフトウェアの一部が、バイパス同期化メカニズム（ETK や CAN バスなど）によりこれら 2 つのプロセッサに分散されます。

バイパスファンクション、つまりラピッドプロトタイピングシステム上で実行されるファンクションの処理は、通常、制御フローインターフェースまたはトリガにより ECU 側から開始されます。バイパスファンクションの出力値については ECU 側で信頼性のチェックが行われます。この場合、ECU とラピッドプロトタイピングシステムは同期的に稼働しますが、別の方法として、トリガを使用しない非同期通信を実装することもできます。

2 つのプロセッサで 1 つのシステムを構成しているので、制御する PC はこの 2 つのプロセッサを「1 つのシステム」として扱う必要があります。つまり PC に両方のプロセッサにアクセスできなければなりません。この状況は INCA によって容易に実現することができます。これは、INCA がラピッドプロトタイピングターゲットと ECU ターゲットの両方にアクセスする機能を持っているためです。

ETAS Experiment Environment でバイパス実験を行う際は、ETAS Experiment Environment から実験ターゲットにアクセスし、バイパスフック変数の測定と適合を行うことができますが、ECU に直接アクセスすることはできません。

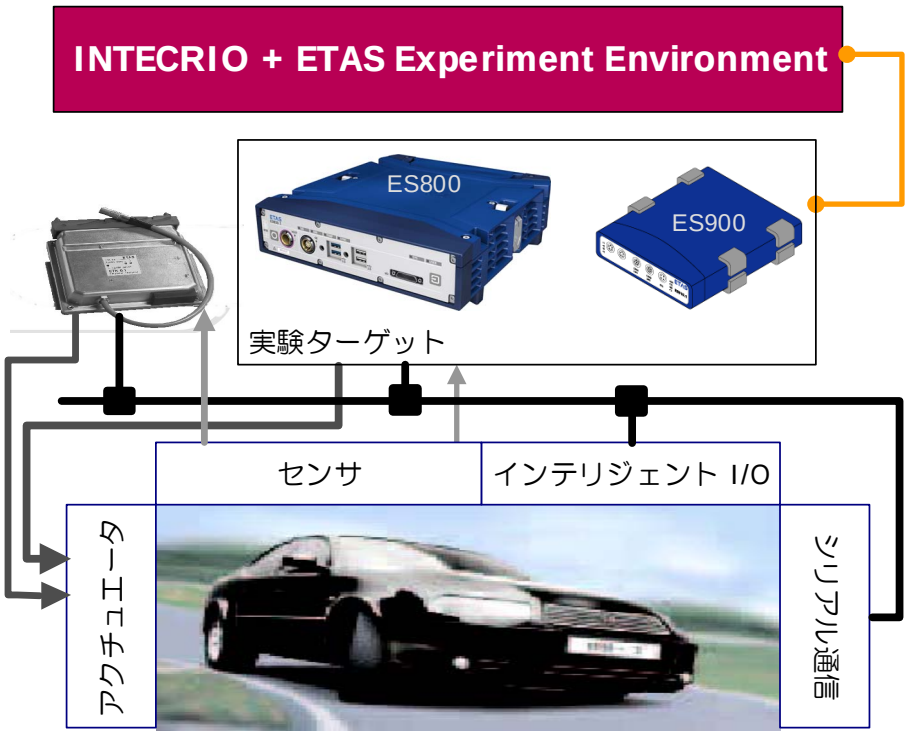


図 5-42 INTECRIO と ETAS Experiment Environment を用いたバイパス実験

5.10.4.2 フルパス実験

ソフトウェアファンクションの評価が完了した ECU やバイパスインターフェースがまだ使用できる状態になっていない場合、または新しいセンサやアクチュエータの検証を行う必要がある場合は、一般的にはフルパス実験の方が適しています。フルパス実験においては、リアルタイム挙動を保証するラピッドプロトタイピングハードウェアが必要で、場合によってはそれを監視する機能も必要です。またファンクションの実行に必要なすべてのセンサ／アクチュエータインターフェースも必要となります。

INTECRIO のフルパス実験では、アプリケーションソフトウェア全体（制御アルゴリズム）がラピッドプロトタイピングハードウェア上で稼働します。ハードウェアは ECU を必要としない「スタンドアロンモード」で稼働し、I/O インターフェースによって外界とのインターフェースが実現され（図 5-43 参照）、ここにセンサとアクチュエータが直接接続されます。

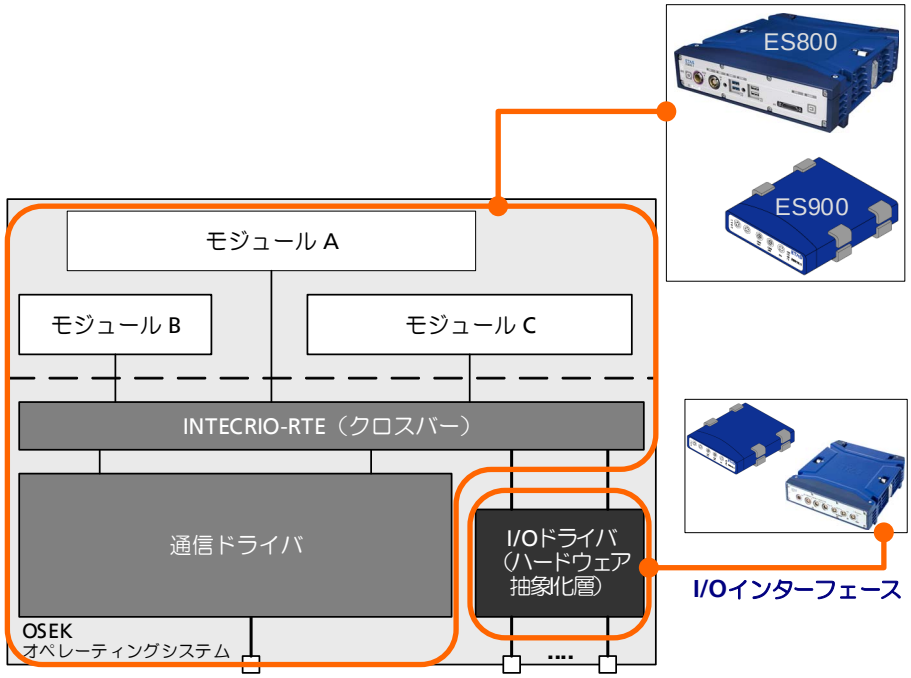


図 5-43 スタンドアロンモードのフルパス実験

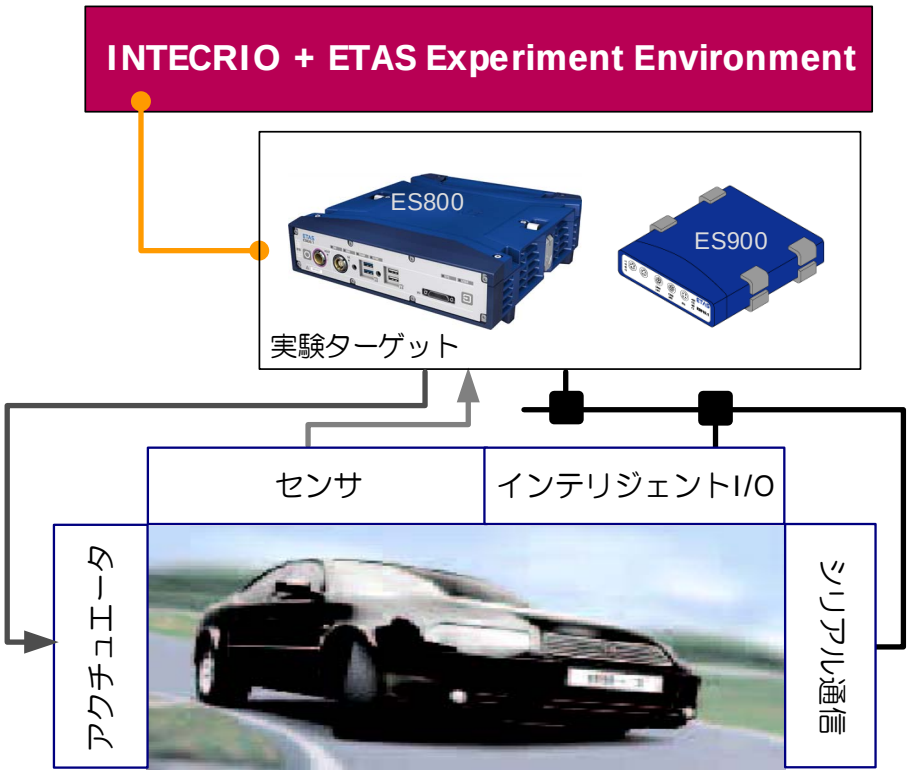


図 5-44 INTECRIO と ETAS Experiment Environment を用いたフルパス実験

なおフルパス実験は、INTECRIO と ETAS Experiment Environment の代わりに、INCA/INCA-EIP を使用して行うこともできます。

5.10.4.3 X パス実験

X パス実験は、バイパス実験とフルパス実験を組み合わせた実験です。ここではバイパスフックを持つ ECU が、ラピッドプロトタイピングハードウェアによって外界とのインターフェースとして利用されます（図 5-45 参照）。

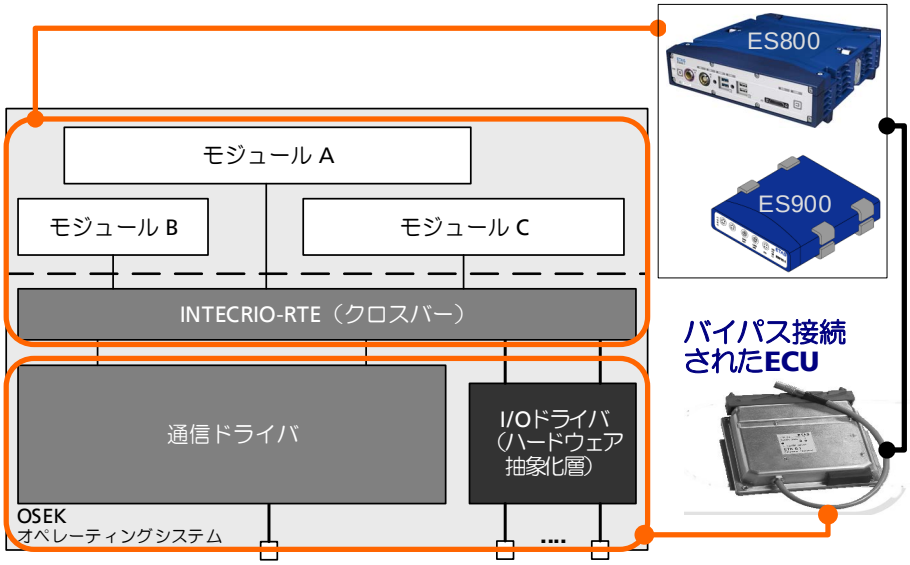


図 5-45 ECU を I/O として利用する X パス実験

5.10.5 ETAS Experiment Environment で行う仮想プロトタイピング実験

ETAS Experiment Environment では、ラピッドプロトタイピング用の機能（変数の参照やパラメータの編集など）をそのまま「仮想プロトタイピング」でも利用することができます。同時に、これらの機能は MATLAB/Simulink や ASCET から直接実行することもできます。さらに、実行中のプロトタイプのパラメータやモジュール接続を変更することにより、さまざまなモデルコンフィギュレーションを比較することができます。

5.11 ドキュメンタ

INTECRIO の「ドキュメンタ」機能により、システムプロジェクトのコンポーネントについてのドキュメントを HTML または PDF フォーマットで生成することができ、これらのドキュメントは、印刷したり他のドキュメントに引用したりして利用できます。

生成されるドキュメントファイルには、システムプロジェクトについての一般情報に加え、ソフトウェア／ハードウェアシステムや OS コンフィギュレーションについての情報も出力されます。出力する内容については、以下のような項目を個別に指定することができます。

システムプロジェクト

- 一般情報（INTECRIO のバージョン、作成日付など）
- ワークスペースの情報
- システムレベルの接続（つまりハードウェアとソフトウェアの接続）

ソフトウェアシステム

- ソフトウェアシステムについての情報（一般情報、ソフトウェアシステムおよびファンクションのシグナルソース／シンク、ソフトウェアシステムレベルの接続など）
- 内包されるモジュールについての情報（一般情報、モジュールのコードとファイルに関する情報、モジュールのシグナルソース／シンク、パラメータ、プロセスなど）

環境システム

- 環境システムについての情報（一般情報、環境システムおよびファンクションのシグナルソース／シンク、環境システムレベルの接続など）
- 内包されるモジュールについての情報（一般情報、モジュールのコードとファイルに関する情報、モジュールのシグナルソース／シンク、パラメータ、プロセスなど）

ハードウェアシステム

- ハードウェアシステムについての情報（一般情報、ECU、CPU、ハードウェアのシグナルソース／シンク、ハードウェアシステムレベルの接続など）
- デバイスについての情報（名前、シグナルグループ、シグナルなど）

OS コンフィギュレーション

- 一般情報
- アプリケーションモードやタスクについての情報
- OS のアクション（プロセスの実行）についての情報
- イベントについての情報

ドキュメント生成に関するオプション設定と、ドキュメント生成方法については、オンラインヘルプに説明されています。

5.12 RTA-TRACE Connectivity

注記

RTA-TRACE は生産中止していますが、既存のインストーラーは現在も使用可能です。

RTA-TRACE Connectivity は ETAS Experiment Environment のアドオンで、これによって RTA-TRACE をラピッドプロトタイピングハードウェアまたは PC（仮想プロトタイピングの場合）に接続することができます。RTA-TRACE を使うと、ターゲット（ES910、ES830、VP-PC）上で実行される実験の時間挙動（VP）またはリアルタイム挙動（RP）をモニタすることができます。

RTA-TRACE の詳細や操作方法については RTA-TRACE のマニュアルをご覧ください。

6 SCOOP と SCOOP-IX

本章では C コードインターフェースディスクリプションの記述や管理を行うためのコンセプトを紹介します。INTECRIO でモデルの統合を行う際は、このインターフェースが非常に重要な役割を果たします。

組み込み制御ソフトウェアにおいては、標準の C コードには以下のようなセマンティックレベルの情報が含まれていないため、ラピッドプロトタイピング用にコードを統合することは容易ではありません。

- 変数の変換式、最小値、最大値、上下限値や、関数の戻り値と引数などの、実装情報
- 不規則に定義された複数の配列や、構造体（`struct`）で定義されている特性値（参照テーブル）のグルーピング情報
- 実験においてどのエレメントが測定または適合に使用されるかを示す情報
- 自動生成されたソースファイルの場合は、モデルのオリジナルデータ（モデル名、物理単位、組み込まれているコンポーネントやブロック、注記、メッセージ／プロセス／シグナル／パラメータの区別）

さらに、以下のような情報も実質的に C コードの中に隠ぺいされてしまうため、容易に抽出することはできません。

- ターゲット固有の `#pragma` 文により記述されたメモリクラス
- `inline` や `far` などのターゲット固有の修飾子により記述された、変数や関数の属性

本章で紹介する「SCOOP」（Source Code, Objects, and Physics）のコンセプトは、これらのインターフェース情報を収集して使用するためのものです。

SCOOP のコンセプトは、インターフェース記述言語（ARXML[1] や、CORBA[2]、Microsoft COM などのインターフェース記述言語に相当します）と、インターフェースディスクリプションの作成や管理を行うツールとで実現されます。

インターフェース記述言語「SCOOP-IX」（6.2 項「SCOOP-IX 言語」を参照してください）は、広い意味での「インターフェース」についてあらゆる情報を詳細に記述するための言語です。SCOOP-IX ディスクリプションは、ツール間のデータ交換に利用したり、オープンな C ソースコードやコンパイル済み C コードの統合処理に利用したりすることができます。

6.1 SCOOP コンセプト

SCOOP のアプローチが目的としているのは、前述のような情報を、コードの実際のインターフェースと共に C コードレベルで統一された「ディスクリプション」として記述することです。SCOOP-IX インターフェースディスクリプションには最初に説明したセマンティック情報の他に、以下の基本的な情報も含まれています。

- 変数の名前、型、サイズ
- 関数の名前、戻り値、およびシグネチャ
- エレメントが含まれるファイル名

これらの情報から、以下のようなインターフェースが記述されます。

1. AUTOSAR XML
2. Object Management Group によって開発された **C**ommon **O**bject **R**equest **B**roker **A**rchitecture（OMG — https://www.omg.org/ を参照してください）

- INTECRIO とそのコンポーネント（特に以下のコンポーネント）の内部インターフェース
  - 実験ターゲットコンフィギュレータ
  - OS コンフィギュレータ
  - プロジェクトコンフィギュレータ
  - プロジェクトインテグレータ
- さまざまなツール間の通信のためのインターフェースや、INTECRIO が ASCET や MATLAB/Simulink とデータ交換を行うためのインターフェース

SCOOP により提供されるインターフェースディスクリプションは、以下のような目的に利用できます。

- ノウハウ保護（IP 保護）のため、オブジェクトファイルやライブラリを、C コードレベル、物理レベル、セマンティックレベルの包括的なインターフェースディスクリプションとして配布する。
- 異なるソフトウェアモジュールのインターフェースの互換性を、C コードレベル（名前、タイプ、シグネチャなど）だけでなく、物理レベル（実装情報、単位）やセマンティックレベル（レコードレイアウト、メッセージとプロセスのどちらとして使用するか）でも確認するため
- 接続用コードやラッパー関数を生成することにより、互いに対応していないモジュール間のインターフェースを調整し、モジュールを統合できるようにする
- OS コンフィギュレータなどのツールにおいて、コードジェネレータにより生成される詳細情報（ASCET のプロセスやメッセージを表すすべての C エレメントなど）を使用できるようにする

6.2 SCOOP-IX 言語

SCOOP-IX は **SCOOP I**nterface E**x**change Language（SCOOP インターフェース交換言語）の略語です。この言語は SCOOP コンセプトの基礎となります。前述のように、C モジュールのインターフェースを SCOOP-IX で記述することにより、C モジュールを INTECRIO に統合することができます。

SCOOP-IX 言語は XML をベースとしているので、INTECRIO、ASCET、Simulink® などのツールでの使用に適しています。

> **注記**
>
> INTECRIO V5.0 は SCOOP-IX の V1.0、V1.1、V1.2、V1.4、V1.5 をサポートしています。

6.2.1 モジュールとインターフェース

SCOOP コンセプトにおいて「モジュール」とは、共通のインターフェースを備えた 1 つのコンパイル単位（通常は C ファイル）、または複数のコンパイル単位を組み合わせたもの（C コードまたはオブジェクトファイルのグループ、ライブラリなど）として扱われます。1 つのモジュールに対して必ず 1 つの SCOOP-IX ファイルが存在します。

グローバルな C 変数や C 関数も、モジュールインターフェースの一部にすることができ、この場合、それらの変数や関数は「インターフェースエレメント」（下記参照）と呼ばれます。

インターフェースエレメントの特徴は、互いにモジュールの境界を越えたアクセス関係や呼び出し関係を持つパートナーであることです。このため、以下のように定義または宣言された変数や関数は、モジュールのインターフェースエレメントに相当するものと見なされます。

- **エクスポートインターフェース**：モジュール内で、`static` 定義ではなくグローバル定義された C 変数や C 関数です。外部接続に使用されるもので、モジュール内で直接内容を見ることができます。モジュール外からアクセスできないエレメントは SCOOP-IX ディスクリプションから除外されます。
- **インポートインターフェース**：モジュール内からアクセス（読み取り、書き込み、アドレス演算）される C 変数、またはモジュール内から直接またはアドレス演算処理により呼び出される C 関数のうち、エレメント自体はモジュール外部で定義されているものです。エレメントは宣言文（MISRA[1] 互換）によって定義されています。

6.2.2 C コードインターフェースのディスクリプション

C コードインターフェースのディスクリプションは、ソースコードに含まれている情報で構成されます。インターフェースエレメントは「変数」と「関数」の 2 種類に分類され、それぞれ固有のインターフェースエレメントが記述されます。

その他、エレメントとモジュールについての以下のような一般情報も SCOOP-IX で扱うことができます。

C 変数のインターフェースディスクリプション（133 ～ 135 ページの 3 つの `<dataElement>` ブロックを参照してください）は基本的に以下の情報で構成されています。

- 変数の名前（134 ページの `<dataCInterface>` ブロックを参照してください）
- 変数の C データ型（134 ページの `<type>` ブロックを参照してください）
- 変数が配列（またはマトリックス）の場合は、x（および y）要素の数
- メモリ配置に関する情報（`extern`、 `static`、 `const` や、`volatile`、 `far`、 `huge`、 `#pragma` 文など）
- 任意指定の初期設定値（134 ページの `<initValue>` ブロックを参照してください）

また C 関数のインターフェースディスクリプション（136 ページの `<functionElement>` ブロックを参照してください）は基本的に以下の情報で構成されています。

- 関数の名前（136 ページの `<functionCInterface>` ブロックを参照してください）
- 関数のメモリ配置に関する情報（`extern`、 `static`（一般関数）、`inline` （ターゲット固有）、 `#pragma` 文など）
- 関数の引数と戻り値についての以下のような情報
  - すべての引数の名前
  - すべての引数と戻り値の C データ型（136 ページの `<return>` ブロックを参照してください。）
  - メモリ配置に関する情報（`const`、 `far`、 `huge` など）

1. Motor Industry Software Reliability Association

– 引数の並び順

C 変数と C 関数については、上記の情報に加えて以下のような一般情報も記述されます。

- インターフェースエレメントのタイプ（インポート／エクスポート、134 ページと 136 ページの `interfaceKind` パラメータを参照してください）
- ファイルオリジン（134 ページと 136 ページの `<fileOrigin>` ブロックを参照してください）

各モジュールのインターフェースディスクリプションには、各エレメント自体の情報以外に以下のような一般情報も記述されます。

- エレメントのファイルオリジン（132 ページの `<fileContainer>` ブロックを参照してください）
- 統合する C ヘッダファイル（同じく `<fileContainer>` ブロックを参照してください）
- モジュールの形態（ソースコード、オブジェクトファイル、またはライブラリ。131 ページの `<constitution mode>` ブロックと `mode` パラメータを参照してください）
- モジュールを実行するハードウェアターゲット（132 ページの `<target>` ブロックを参照してください）とコンパイラ（132 ページの `<tool>` ブロックを参照してください）
- コンパイラオプションなどの設定（132 ページの `<configuration>` ブロックを参照してください）

6.2.3 セマンティックス情報のディスクリプション

セマンティックス情報は、C コードインターフェースの情報のように分析を行ってソースコードから抽出することはできません。代わりに、マニュアル操作で、またはコードジェネレータによるコード生成時に自動生成する必要があります。ASCET や MATLAB/Simulink の接続機能により後者の方法が可能になります。

エレメントのセマンティックス情報は「モデルオリジン」、「インプリメンテーション」、および「使用」に分類されます。これらの分類については以降の項で説明し、さらにその他のモジュール固有の情報についても説明します（124 ページ「モジュールデータ」参照）。

6.2.3.1 モデルオリジン

対象となる C コードが自動生成された場合、または形式モデル（ブロックダイアグラム、ステートマシン、制御／データフローダイアグラムなど）に基づいてハンドコーティングを行った場合は、各インターフェースエレメントのオリジンをモデル内に記述しておくことができます。

この「モデルオリジン」の主な使用目的は、ドキュメンテーションや、ユーザーがモデルを把握しやすくすることなどがあげられます。この情報はプロジェクトコンフィギュレータなどの設定ツールで表示できるので、この情報をもとにモデルエレメントを検索することができます。

さらに、この情報を利用して、特定の情報を他のツールに転送することもできます。たとえば、プロセスやメッセージのために生成された C エレメントを ASCET でマークすると、OS コンフィギュレータはインターフェースディスクリプション全体の中からそのエレメントを正確に特定し、オペレーティングシステム設定用の候補としてユーザーに提示することができます。

さらに、オリジンの記述を行うと、モデルの最高レベルについて、ドメインやツールの境界を越えた整合性や妥当性に関するセマンティックスチェックを行えるようになります。

モデルオリジンについての情報は一般情報とモデル固有情報に分けられます。後者については ASCET および MATLAB/Simulink の例を用いて説明します。

モデルオリジンについての一般情報（134 ページと 136 ページの `<modelOrigin>` ブロックを参照してください）には以下のエレメントがあります。

- モデルの名前。つまり、インターフェースエレメントの表示名（134 ページと 136 ページの `<name>` ブロックを参照してください）、またはメモリエレメント、シグナル、メソッドなどのエレメント名などです。
- モデル内でユニークな識別子（134 ページと 136 ページの `identifier` オプションを参照してください）
- 階層的なモデル構造内のパス（134 ページと 136 ページの `<modelLink>` ブロックを参照してください）
- モデル型。C 変数、関数の戻り値／引数などに使用される `continuous`、 `discrete`、 `Boolean`、 `array` など（134 ページの `<modelType>` ブロックを参照してください）
- モデルの形態。C 変数の場合は変数、パラメータ、定数など（134 ページと 136 ページの `<modelKind>` ブロックの `kind` オプションを参照してください）
- 他のモデルからの可視性。`public` または `private` のいずれか（134 ページと 136 ページの `visibility` オプションを参照してください。）

  `public` は ASCET のスコープ exported に相当し、`private` は local に相当します。
- C 変数の論理フロー方向。入力ポートや出力ポートなど（133 ページの `<flowDirection>` ブロックを参照してください）
- C 変数と、関数の戻り値／引数の物理単位
- C 変数と、関数の戻り値／引数のモデルレベルの値の範囲
- 注記のテキスト。ユーザーコメントやその他の一般的なモデル情報など（134 ページと 136 ページの `<annotation>` ブロックを参照してください

使用される BMT（挙動モデリングツール）によっては、SCOOP-IX モジュールの統合には所定の「ドメイン固有情報」が必要となります。

以下の情報は、ASCET で生成されたコードについて特に重要なものです。

- 対応するインターフェースエレメントが組み込まれている ASCET コンポーネント
- コンポーネントのタイプ（クラス、モジュール、プロジェクト。134 ページの `<pathNode>` ブロック、 `kind="asd:module"` オプションを参照してください）
- インターフェースエレメントのタイプ（エレメント、メッセージ、リソース、メソッド、プロセス、タスク。134 ページの `<pathNode>` ブロック、 `kind="asd:element"` オプションを参照してください）

RTA-OSEK などのリアルタイムオペレーティングシステムのプロセスは、C コードレベルにおいては `void/void` 関数で表現されます。OS 設定とプロジェクト統合の際には、特に以下の情報が重要となります。

- プロセス内でアクセスされるメッセージ、またはプロセス内で呼び出される C 関数内でアクセスされるメッセージ（136 ページの <messageAccess> ブロックを参照してください）、およびそれらのアクセスモード（send、receive。136 ページの send オプションを参照してください）
- プロセス内でアクセスされるリソース、またはプロセス内で呼び出される C 関数内でアクセスされるリソース（136 ページの <resourceAccess> ブロックを参照してください）
- 以下のような、モデルレベルの時間的要件（136 ページの <constraint> ブロックを参照してください）
  - 実行周期（136 ページの <period> ブロックを参照してください）、オフセット、デッドライン、優先度（136 ページの priority オプションを参照してください）
  - トリガタイプ（初期化／タイマ／割り込み／ソフトウェア。136 ページの trigger オプションを参照してください）
  - スケジューリングモード（プリエンプティブ、協調つまり非プリエンプティブ。136 ページの <scheduling> ブロックを参照してください）

  実際の優先度レベルはオペレーティングシステムと CPU により異なるので、background、 low、 normal、 high、 scheduler などのような不明確な定義を使用できます。

以下の情報は、MATLAB/Simulink で生成されたコードについて特に重要なものです。

- 各インターフェースエレメントに対応するアイテムを含む MATLAB/Simulink のサブシステムまたはブロック、およびそのタイプ
- インターフェースエレメントに対応するアイテムのタイプ（シグナルまたはパラメータ）

MATLAB/Simulink モデルのスキャンレートは、定義済みの実行時間制限など、比較できるものを用いて決定し、定義することができます。

6.2.3.2 インプリメンテーション

データの整合性チェックや接続用コードの生成を行えるようにするためには、C コードインターフェースの実装に関する情報（「インプリメンテーション」）が必要です。インプリメンテーションは、C 変数、および関数の戻り値と引数について定義されます。

インプリメンテーション情報（134 ページの <implementation> ブロックを参照してください）には以下の情報が含まれています。

- モデルエレメントのデータと、そのモデルエレメントに対応する C コードのデータとの関係を記述する変換式（133 ページの <conversion> ブロックと 134 ページの <conversionRef> ブロックを参照してください）
- C コードレベルの最小値と最大値（134 ページの <valueRange> ブロックを参照してください）
- 値の範囲の限界を超えた場合のリミッタの使用と、（SCOOP-IX V1.2 の場合の）解決スキーム（134 ページの <saturation> ブロックを参照してください）

  <saturation> ブロックにはオプションの value、resolution、assignment が含まれます。ASCET インプリメンテーションエディタでの設定に応じて、オプションは以下のようにセットされます。

- *Limit to maximum bit length* がオン（オフ）の場合、`value` は `true`（`false`） にセットされます。
- *Limit to maximum bit length の隣のコンボボックスの設定に応じて*、`resolution` は `automatic`、 `keep`、 `reduce` のいずれかにセットされます。
- *Limit to maximum bit length* がオン（オフ）の場合、`assignment` は `true` （`false`） にセットされます。

Simulink モデルの場合、またはリミッタが使用されていない ASCET モデルの場合、`<saturation>` ブロックは省略されます。

- ASCET V6.4 で生成された SCOOP-IX ファイルの場合：`<zeroExcluded>` ブロック（135 ページ参照）の内容は常に `value="false"` となります。

  ASCET V5.0 ～ V6.3 で生成された SCOOP-IX ファイルの場合：`<zeroExcluded>` ブロックには、インターバル内のゼロが明示的に除外されるかどうかについての情報が含まれます。

6.2.3.3 使用

インターフェースエレメントの使用については以下のセマンティックス情報が重要です（データエレメントの場合のみ）。

- 特性値と、その特性値の元となる C 変数との関係、および特性値についての情報（座標軸、サイズ、レコードレイアウトなど）
- 測定変数と適合変数のどちらとして扱われるか：`<usage>` ブロック（測定— 135 ページ、適合— 135 ページ）
- 実験ターゲットのシミュレーションインターフェース用の疑似アドレス、またはビットマスクを用いた ECU アドレス

6.2.3.4 モジュールデータ

モジュールの SCOOP-IX ディスクリプション（130 ページの `<moduleInfo>` ブロックを参照してください）には以下のアドオン情報が含まれる可能性があります。

- モジュール名（130 ページの `<name>` ブロックを参照してください）
- インターフェースディスクリプションの内容のモジュールバージョン（130 ページの `<version>` ブロックを参照してください）
- モジュールを作成した日付と時刻（130 ページの `<dateTime>` ブロック、`kind="created"` オプションを参照してください）
- 最後に変更を行った日付と時刻（130 ページの `<dateTime>` ブロック、`kind="lastModified"` オプションを参照してください）
- インターフェースディスクリプションの完成度（130 ページの `<completion>` ブロックを参照してください）。以下のいずれかの値です。
  - `basic`（C コードだけのインターフェースデータで、セマンティックス情報が含まれません）
  - `in progress`（セマンティックス情報が部分的に記述された、中間レベル）
  - `full`（ドキュメントは完成していると見なされます。ただしセマンティックス情報は完全である必要はありません。）
- ユーザーと会社についての情報（131 ページの `<company>`、`<user>`、`<creators>` ブロックを参照してください）

- モデル生成に使用された BMT の名前とバージョン（131 ページの `<tool>` ブロックを参照してください）
- 注記のテキスト（131 ページの `<annotation>` ブロックを参照してください）

6.2.4 参照モデル

V5.0.2 より INTECRIO は、Simulink の「モデル参照」の機能をサポートしています。参照モデルを含む Simulink モデル用のコードを生成すると、以下のようなことが行われます。

- メインモデル用に、SCOOP-IX V1.5 の `*.six` ファイルが生成されます。その内容は、6.2.4.1 項「`*.six` ファイルからの抜粋」（125 ページ）を参照してください。

  この `*.six` ファイルには、すべての参照モデル（メインモデルから参照されるモデルと他の参照モデルから参照されるモデル）へのリンクが含まれます。

  モデルを INTECRIO にインポートする際に選択するのは、この `*.six` ファイルだけです。
- 各参照モデルについて、*`<referenced_model_name>`*`.ref_six` という名前の SCOOP-IX V1.5 のファイルが生成されます。その内容は、6.2.4.2 項「`*.ref_six` ファイルからの抜粋」（127 ページ）を参照してください。
- `*.ref_six` ファイルには、他の `*.six` ファイルや `*.ref_six` ファイルへのリンクは含まれません。参照モデルに他の参照モデルが含まれる場合は、メインの `*.six` ファイル内でリンクされます。

6.2.4.1 `*.six` ファイルからの抜粋

以下の例は、参照モデルを含む Simulink モデルのメインモデルの `*.six` ファイルから抜粋したものです。

メインモデルへの参照と参照モデルへの参照は、青い文字で示されています。

```
...
<module
   xmlns="http://www.etas.com/scoop-ix/1.5"
   xmlns:ix="http://www.etas.com/scoop-ix/1.5"
   xmlns:mlsl="http://www.etas.com/scoop-ix/1.5/    ↵
      modelDomain/matlab-simulink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.etas.com/scoop-ix/1.5 ↵
      file://C:\ETAS\INTECRIO4.7\SCOOP-IX\1.2/schemas/ ↵
      scoop-ix-domain-mlsl.xsd"
   xmlns:html="http://www.w3.org/1999/xhtml">
   <directoryLocations scheme="MATLAB 9.3">
      ...
   </directoryLocations>
   <moduleInfo identifier="MyMainModel">
      <name>MyMainModel</name>
      <modelLink href="mlsl://&amp;quot;{{modelDir}}    ↵
         MyMainModel.slx&amp;quot;"/>
      <version major="1" minor="9"/>
      ...
      <creators>
         <user lastName="MOL9FE" role="creator"/>
         <tool kind="environment" vendor="The Mathworks, ↵
            Inc." name="MATLAB">
```

```
      ...
    </tool>
    <tool kind="modeler" vendor="The Mathworks, Inc." ↵
      name="Simulink">
      ...
    </tool>
    <tool kind="modeler" vendor="The Mathworks, Inc." ↵
      name="Stateflow">
    </tool>
    <tool kind="codeGenerator" vendor="The Mathworks, ↵
      Inc." name="Simulink Coder">
      ...
    </tool>
    <tool kind="codeGenerator" vendor="The Mathworks, ↵
      Inc." name="MATLAB Coder">
      ...
    </tool>
    <tool kind="codeGenerator" vendor="ETAS GmbH" ↵
      name="Connector for Simulink (IRT)"      ↵
      family="INTECRIO Tool Suite">
      <version major="4" minor="7" year="2019" ↵
        month="1" day="1"/>
      <configuration>
        <option identifier="SCOOPIXFileName"> ↵
          <![CDATA[MyMainModel.six]]></option>
        ...
      </configuration>
    </tool>
  </creators>
</moduleInfo>

<codeInfo>
  <constitution mode="sourceCode"/>
  <dateTime kind="created" year="2019" month="5" ↵
    day="16" hour="11" minute="20" second="37"/>
  <target>
    <board vendor="ETAS GmbH" model="INTECRIO  ↵
      Generic Experimental Target"/>
    <tool kind="compiler" vendor="GNU Project" ↵
      family="GNU Compiler Collection"         ↵
      name="GNU C Compiler">
      <configuration>
        <optionKind name="macroDefine" prefix="-D"/>
        <optionKind name="includeDirectory"    ↵
          prefix="-I"/>
        <!-- RTW specific defines -->
        <option kind="macroDefine" name="USE_RTMODEL"/>
        <option kind="macroDefine" name="MODEL"> ↵
          MyMainModel</option>
        ...
        <!-- RTW specific include directories -->
        <option kind="includeDirectory">       ↵
          <![CDATA[{{codeDir}}]]></option>
        ...
        <!-- S-Function specific include directories -->
        <!-- Referenced models local include
          directory -->
        <option kind="includeDirectory">       ¿
          <![CDATA[{{codeDir}}                 ¿
          referenced_model_includes]]></option>
      </configuration>
```

```
        </tool>
      </target>
    </codeInfo>
    <!-- Model specific files -->
    <fileContainer constitution="sourceCode">
      <pathBase path="{{codeDir}}"/>
      <!-- Model specific source files -->
      <file name="MyMainModel_types.h" kind="header"/>
      <file name="MyMainModel.h" kind="header"/>
      <!-- used through rtwShared.lib:              ↵
        zero_crossing_types.h -->
      <file name="MyMainModel.c" kind="body"/>
      <file name="MyMainModel_private.h" kind="header"/>
      <file name="rtmodel.h" kind="header"/>
      <!-- used through rtwShared.lib: rtGetInf.h -->
      <!-- used through rtwShared.lib: rtGetInf.c -->
      <!-- used through rtwShared.lib: rtGetNaN.h -->
      <!-- used through rtwShared.lib: rtGetNaN.c -->
      <file name="MyMainModel_main.c" kind="body"/>
      <!-- Additionally registered model specific source
        files -->
      <file name="rt_sim.c"                          ↵
        path="{{codeDir}}external\rtw\c\src\" kind="body"/>
      <!-- Target specific libraries -->
      <file name="rtwStaticLib.lib" kind="symbolicLibrary"/>
      <file name="rtwSharedLib.lib" kind="symbolicLibrary"/>
      <!-- Additional files -->
    </fileContainer>
    <fileContainer constitution="referencedModels">
      <!-- SCOOP-IX files for referenced models -->
      <file name="MySub.ref_six"                     ↵
        path="D:\ETASData\INTECRIO4.7\User\ModelRef\ ↵
        modules\mysub\" kind="SIX" format="SCOOP-IX"/>
    </fileContainer>
...
```

> **注記**
>
> メインモデルの *.six ファイル内の **<fileContainer constitution="referencedModels">** セクションには、すべての参照モデル（メインモデルから参照されるモデルと他の参照モデルから参照されるモデル）へのリンクが含まれます。

6.2.4.2 *.ref_six ファイルからの抜粋

以下の例は、参照モデルの *.ref_six ファイルから抜粋したものです。

モデルへの参照は、青い文字で示されています。

```
...
<module
  xmlns="http://www.etas.com/scoop-ix/1.5"
  xmlns:ix="http://www.etas.com/scoop-ix/1.5"
  xmlns:mlsl="http://www.etas.com/scoop-ix/1.5/    ↵
    modelDomain/matlab-simulink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.etas.com/scoop-ix/1.5 ↵
    file://C:\ETAS\INTECRIO4.7\SCOOP-IX\1.2/schemas/ ↵
    scoop-ix-domain-mlsl.xsd"
  xmlns:html="http://www.w3.org/1999/xhtml">
```

```
<directoryLocations scheme="MATLAB 9.3">
   ...
</directoryLocations>
<moduleInfo identifier="MySub">
   <name>MySub</name>
   <modelLink                                          ↵
     href="mlsl://&amp;quot;{{modelDir}}MySub.slx&amp;quot;"/>
   <version major="1" minor="26"/>
   ...
   <creators>
      <user lastName="MOL9FE" role="creator"/>
      <tool kind="environment" vendor="The Mathworks, ↵
        Inc." name="MATLAB">
        ...
      </tool>
      <tool kind="modeler" vendor="The Mathworks, Inc." ↵
        name="Simulink">
        ...
      </tool>
      <tool kind="modeler" vendor="The Mathworks, Inc." ↵
        name="Stateflow">
      </tool>
      <tool kind="codeGenerator" vendor="The Mathworks, ↵
        Inc." name="Simulink Coder">
        ...
      </tool>
      <tool kind="codeGenerator" vendor="The Mathworks, ↵
        Inc." name="MATLAB Coder">
      </tool>
      <tool kind="codeGenerator" vendor="ETAS GmbH" ↵
        name="Connector for Simulink (IRT)"      ↵
        family="INTECRIO Tool Suite">
        <version major="4" minor="7" year="2019"  ↵
          month="1" day="1"/>
        <configuration>
          <option identifier="SCOOPIXFileName"> ↵
            <![CDATA[MySub.ref_six]]></option>
          ...
        </configuration>
      </tool>
   </creators>
</moduleInfo>

<codeInfo>
   <constitution mode="sourceCode"/>
   <dateTime kind="created" year="2019" month="5" ↵
     day="16" hour="11" minute="20" second="19"/>
   <target>
      <board vendor="ETAS GmbH" model="INTECRIO Generic ↵
        Experimental Target"/>
      <tool kind="compiler" vendor="GNU Project" ↵
        family="GNU Compiler Collection"         ↵
        name="GNU C Compiler">
        <configuration>
          <optionKind name="macroDefine" prefix="-D"/>
          <optionKind name="includeDirectory"    ↵
            prefix="-I"/>
          <!-- RTW specific defines -->
          <option kind="macroDefine" name="USE_RTMODEL"/>
          <option kind="macroDefine" name="MODEL"> ↵
            MySub</option>
```

```
            ...
            <!-- RTW specific include directories -->
            <option kind="includeDirectory">      ↵
              <![CDATA[{{codeDir}}]]></option>
            ...
            <!-- S-Function specific include directories -->
            <!-- Referenced models local include
              directory -->[1]
              ...
          </configuration>
        </tool>
      </target>
    </codeInfo>
    <!-- Model specific files -->
    <fileContainer constitution="sourceCode">
      <pathBase path="{{codeDir}}"/>
      <!-- Model specific source files -->
      <file name="MySub_types.h" kind="header"/>
      <file name="MySub.h" kind="header"/>
      <!-- used through rtwShared.lib: rtwtypes.h -->
      <!-- used through rtwShared.lib: multiword_types.h -->
      <file name="MySub.c" kind="body"/>
      <file name="MySub_private.h" kind="header"/>
      <!-- Additionally registered model specific  ↵
        source files -->
      <file name="rt_sim.c"                         ↵
        path="{{codeDir}}external\rtw\c\src\" kind="body"/>
      <!-- Target specific libraries -->
      <file name="rtwStaticLib.lib" kind="symbolicLibrary"/>
      <file name="rtwSharedLib.lib" kind="symbolicLibrary"/>
      <!-- Additional files -->
    </fileContainer>
    <fileContainer constitution="referencedModels">
    </fileContainer>
...
```

注記

*.ref_six ファイルの <fileContainer constitution="referencedModels"> セクションは、参照モデルに別の参照モデルが含まれている場合であっても、常に空です。

参照モデルへのリンクは、すべてメインモデルの *.six ファイルに含まれます。

6.3 SCOOP-IX の作成と例

INTECRIO に含まれるツールまたは INTECRIO と連結しているツールで統合用 C コードが生成される際は、SCOOP-IX ディスクリプションが生成されます。モデルを ASCET や MATLAB/Simulink で作成する場合は、SCOOP-IX はそれぞれの接続機能（Connectivity）により生成されます。

1. ここには、参照モデルに他の参照モデルが含まれる場合のみ、その情報が含まれます。

ASCET で作成される単純な SCOOP-IX ファイルの例を以下に紹介します。この例はインターフェースディスクリプションを SCOOP-IX で記述する方法を示すものであるため、内容の妥当性については保証されていません。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE module [
   <!ENTITY szlig "&#223;">
   <!ENTITY copy "&#169;">
   <!ENTITY baseTypes-asd SYSTEM                    ↵
      'c:\ETAS\ASCET6.4\Formats\SCOOP-IX\1.2\common\ ↵
        baseTypes-asd.xml'>
   ]>
<?xml-stylesheet type="text/xsl" href="c:\ETAS\    ↵
   ASCET6.4\Formats\SCOOP-IX\1.2\common\           ↵
   showSCOOP-IX.xsl"?>
<!--
<h1>SCOOP-IX</h1>
<p>
   <strong>Copyright &copy; 2002-2004 ETAS GmbH</strong>,↵
      Borsigstra&szlig;e 14, D-70469 Stuttgart.    ↵
      <em>All rights reserved.</em>
</p>
-->
<module
   xmlns="http://www.etas.de/scoop-ix/1.2"
   xmlns:ix="http://www.etas.de/scoop-ix/1.2"
   xmlns:asd="http://www.etas.de/scoop-ix/1.2/     ↵
      modelDomain/ascet"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.etas.de/scoop-ix/1.2 ↵
      c:\ETAS\ASCET6.4\Formats\SCOOP-IX\1.2\Schemas\ ↵
        scoop-ix-domain-asd.xsd"
   xmlns:html="http://www.w3.org/1999/xhtml" >

   <directoryLocations scheme="ASCET 6.4">
      <directory identifier="integratorDir"        ↵
        path="E:\ETAS\INTECRIO5.0\" ></directory>
      <directory identifier="toolDir"              ↵
        path="c:\ETAS\ASCET6.4\" ></directory>
      <directory identifier="modelDir"             ↵
        path="c:\ETASData\ASCET6.4\Database\INTECRIO\" >↵
        </directory>
      <directory identifier="codeDir"              ↵
        path="c:\ETASData\ASCET6.4\Database\INTECRIO\ ↵
        Project\CGen\" ></directory>
   </directoryLocations>

   <moduleInfo identifier=                         ↵
      "_040VSM3H60001KO7102G5GFA1O5G0">
      <name>ASDSimpleModel</name>
      <modelLink href="asd://{{modelDir}}?Training/ ↵
        ASDSimpleModel" ></modelLink>
      <version major="1" minor="0" ></version>
      <dateTime kind="created" year="2011" month="02" ↵
        day="13" hour="15" minute="37" second="04" > ↵
        </dateTime>
      <dateTime kind="lastModified" year="2011"    ↵
        month="03" day="04" hour="17" minute="14"  ↵
        second="44" > </dateTime>
      <completion degree="full" ></completion>
      <suitability>
```

```
      <application domain="rapidPrototyping"      ↵
        addressesAvailable="true"                 ↵
        instanceTreeRootIdentifier=               ↵
          "_040VSM3H60001KO7102G5GFA1O5G0instance" ↵
        setGetDeltaTIdentifier=                   ↵
          "__040VSM3H60001KO7102G5GFA1O5G0">      ↵
        </application>
    </suitability>
    <company name="ETAS GmbH" department="ETAS/PAC-F1" ↵
      city="Stuttgart" country="Germany" />
    <user lastName="Doe" firstName="John" title="Dr" > ↵
      </user>
    <creators>
      <user lastName="Doe" firstName="John"       ↵
        title="Dr" ></user>
      <tool kind="modeler" vendor="ETAS GmbH"     ↵
        name="ASCET">
        <version major="6" minor="1" revision="1" > ↵
          </version>
        <configuration >
          <option identifier="ignoreInternalMessages" ↵
            > false</option>
        </configuration>
      </tool>
      <tool kind="codeGenerator" vendor="ETAS GmbH" ↵
        name="ASCET">
        <version major="6" minor="1" revision="1" > ↵
          </version>
        <mode name="experiment"                   ↵
          value="Implementation" ></mode>
        <configuration>
          <option identifier="Code Generator" > ↵
            Implementation Experiment</option>
          <option identifier="Target" >Prototyping ↵
            </option>
          ...[1]
        </configuration>
      </tool>
    </creators>
    <annotation>
      <ix:documentation xmlns="http://www.w3.org/ ↵
        1999/xhtml">
        <p>This is a sample module interface     ↵
          description file. It is used for       ↵
          demonstrating an interface
        description in the <b>SCOOP-IX</b>        ↵
          language.</p>
        <p>Neither is its content supposed to make ↵
          any sense at all, nor has its correctness ↵
          been checked by compilation.</p>
      </ix:documentation>
    </annotation>
  </moduleInfo>

  <codeInfo>
    <constitution mode="sourceCode" ></constitution>
```

1. SCOOP-IX ファイルにはプロジェクト属性のすべての設定が含まれますが、ここには最初の 2 つだけが記載されています。

```
   <dateTime kind="created" year="2011" month="02" ↵
      day="13" hour="15" minute="38" second="4" > ↵
      </dateTime>
   <target>
      <processor vendor="Motorola" model="MPC750" > ↵
         </processor>
      <board vendor="ETAS GmbH" model="Prototyping" > ↵
         </board>
      <tool kind="compiler" vendor="GNU Project"  ↵
         family="GNU Compiler Collection"        ↵
         name="GNU C Compiler for Embedded PowerPC  ↵
         target">
         <configuration>
            <optionKind name="macroDefine" prefix="-D" > ↵
               </optionKind>
            <optionKind name="includeDirectory"   ↵
               prefix="-I" ></optionKind>

            <!-- ASCET specific defines -->
            <option kind="macroDefine"            ↵
               name="EXT_INTEGRATION" ></optionKind>

            <!-- ASCET specific include directories -->
            <option kind="includeDirectory">      ↵
               <![CDATA[{{codeDir}}]]></option>
         </configuration>
      </tool>
   </target>
   <target >
      <board vendor="ETAS GmbH" model="INTECRIO  ↵
         Generic Experimental Target" >
      </board>
      <tool kind="compiler" vendor="GNU Project" ↵
         family="GNU Compiler Collection"        ↵
         name="GNU C Compiler" >
         <configuration >
            <optionKind name="macroDefine" prefix="-D" > ↵
               </optionKind>
            <optionKind name="includeDirectory"   ↵
               prefix="-I" ></optionKind>
            <option kind="macroDefine" name=      ↵
               "EXT_INTEGRATION" ></option>
            <option kind="includeDirectory" >     ↵
               <![CDATA[{{codeDir}}]]></option>
         </configuration>
      </tool>
   </target>
</codeInfo>

<fileContainer complete="false">
   <pathBase path="{{codeDir}}" ></pathBase>
   <!-- model specific C files -->
   <file name="_asd_pid.c" kind="body" ></file>
   <file name="asdsmpm.c" kind="body" ></file>
   <file name="conf.c" kind="body" ></file>
   <file name="modulem.c" kind="body" ></file>
   <file name="asdsmpm.h" kind="header" ></file>
   <file name="conf.h" kind="header" ></file>
   <file name="modulem.h" kind="header" ></file>
   <!-- additional files -->
```

```
   <file name="ASDSimpleModel.a2l"
      content="dataDescription" format="ASAM-2MC" ↵
      formatVersion="1.5" ></file>
</fileContainer>

<interface>
   <modelLinkBase href="asd://
      {{modelDir}}?Training/ASDSimpleModel/" >    ↵
      </modelLinkBase>

   <pathBase path="{{codeDir}}" ></pathBase>
   <headerFile name="asdsmpm.h" ></headerFile>
   <headerFile name="conf.h" ></headerFile>
   <headerFile name="modulem.h" ></headerFile>
   <usage layoutFamily="asd:standardLayout" ></usage>
   &baseTypes-asd;

   <definitions>
      <conversion name="ident">
         <rationalFunction>
            <numerator bx="1" ></numerator>
            <denominator f="1" ></denominator>
         </rationalFunction>
      </conversion>
   </definitions>

   <dataElement interfaceKind="export">
      <dataCInterface identifier=                 ↵
         "MODULE_IMPL_ClassObj.Out1->val">
         <type><typeRef name="real64" ></typeRef></type>
         <fileOrigin name="MODULEM.c" ></fileOrigin>
         <initValue value="0.0" ></initValue>
      </dataCInterface>
      <modelOrigin identifier="ASDSimpleModel.    ↵
         Module.Out1">
         <name>Out1</name>
         <modelLink href="Module.Out1" ></modelLink>
         <modelLocation>
            <pathNode name="Module" kind="asd:module">
               <pathParameter name="asd:implementation" ↵
                  value="Impl" ></pathParameter>
               <pathParameter name="asd:dataSet"    ↵
                  value="Data" ></pathParameter>
            </pathNode>
            <pathNode name="Out1" kind="asd:element" > ↵
               </pathNode>
         </modelLocation>
         <modelKind kind="message" visibility="public">
            <flowDirection in="false" out="true" > ↵
               </flowDirection>
         </modelKind>
         <modelType type="continuous" ></modelType>
         <annotation>
            <ix:documentation xmlns=                 ↵
               "http://www.w3.org/1999/xhtml"        ↵
               lang="en-US">
               This is output message <i>Out1</i> of ↵
                  continuous type.
            </ix:documentation>
         </annotation>
      </modelOrigin>
```

```
    <implementation>
      <conversionRef name="ident" ></conversionRef>
      <valueRange min="-2147483648"                ↵
        max="2147483647" ></valueRange>
      <saturation value="true" resolution="reduce" ↵
        assignment="true" ></saturation>
      <zeroExcluded value="false" ></zeroExcluded>
    </implementation>
    <usage measurement="true" virtual="false"    ↵
      variant="false" >
      <address kind="pseudo" >
        <BLOB kind="KP_BLOB" device="E_TARGET" > ↵
          <![CDATA[2 1001 1 1001 1]]></BLOB>
      </address>
    </usage>
  </dataElement>

  <dataElement interfaceKind="export">
    <dataCInterface identifier=                  ↵
      "ASDSIMPLEMODEL_IMPL_ClassObj.Module->     ↵
      myProduct->val">
      <type><typeRef name="sint32" ></typeRef></type>
      <fileOrigin name="MODULEM.c" ></fileOrigin>
      <initValue value="0" ></initValue>
    </dataCInterface>
    <modelOrigin identifier=
      "ASDSimpleModel.Module.myProduct">
      <name>myProduct</name>
      <modelLink href="ASDSimpleModel.Module.    ↵
        myProduct" ></modelLink>
      <modelLocation>
        <pathNode name="Module" kind="asd:module">
          <pathParameter name="asd:implementation" ↵
            value="Impl" ></pathParameter>
          <pathParameter name="asd:dataSet"      ↵
            value="Data" ></pathParameter>
        </pathNode>
        <pathNode name="myProduct"               ↵
          kind="asd:element" ></pathNode>
      </modelLocation>
      <modelKind kind="variable"                 ↵
        visibility="private" ></modelKind>
      <modelType type="continuous" >
        <valueRange min="-2147483648.0"          ↵
          max="2147483647.0" ></valueRange>
      </modelType>
      <annotation>
        <ix:documentation xmlns=                 ↵
          "http://www.w3.org/1999/xhtml"         ↵
          lang="en-US">
          This is variable <i>myProduct</i> of   ↵
            continuous type.
        </ix:documentation>
      </annotation>
    </modelOrigin>
    <implementation>
      <conversionRef name="ident" ></conversionRef>
      <valueRange min="-2147483648"              ↵
        max="2147483647"> </valueRange>
      <saturation value="true" resolution="reduce" ↵
        assignment="true" ></saturation>
```

```
      <zeroExcluded value="false" ></zeroExcluded>
    </implementation>
    <usage measurement="true" virtual="false"  ↵
      variant="false" >
      <address kind="pseudo" >
        <BLOB kind="KP_BLOB" device="E_TARGET" > ↵
          <![CDATA[2 1001 1 1000 0]]></BLOB>
      </address>
    </usage>
  </dataElement>

  <dataElement interfaceKind="export">
    <dataCInterface identifier=                ↵
      "ASDSIMPLEMODEL_IMPL_ClassObj.Module->   ↵
        myPar->val">
      <type><typeRef name="real64" ></typeRef></type>
      <fileOrigin name="MODULEM.c" lines="23" > ↵
        </fileOrigin>
      <initValue value="3.2" />
    </dataCInterface>
    <modelOrigin identifier="ASDSimpleModel.   ↵
      Module.myPar">
      <name>myPar</name>
      <modelLink href="ASDSimpleModel.Module.myPar" ↵
        > </modelLink>                         ↵
      <modelLocation>
        <pathNode name="Module" kind="asd:module">
          <pathParameter name=
            "asd:implementation" value="Impl" > ↵
            </pathParameter>
          <pathParameter name=                 ↵
            "asd:dataSet" value="Data" >       ↵
            </pathParameter>
        </pathNode>
        <pathNode name="myPar"
          kind="asd:element" ></pathNode>
      </modelLocation>
      <modelKind kind="parameter"              ↵
        visibility="private" ></modelKind>
      <modelType type="continuous" ></modelType>
      <annotation>
        <ix:documentation xmlns=               ↵
          "http://www.w3.org/1999/xhtml"       ↵
          lang="en-US">
          This is parameter <i>myPar</i> of    ↵
            continuous type.
        </ix:documentation>
      </annotation>
    </modelOrigin>
    <implementation>
      <conversionRef name="ident" ></conversionRef>
      <valueRange min="-1.e+037" max="1.e+037" > ↵
        </valueRange>
      <zeroExcluded value="false" ></zeroExcluded>
    </implementation>
    <usage calibration="true" virtual="false"  ↵
      variant="false" >
      <address kind="pseudo" >
        <BLOB kind="KP_BLOB" device="E_TARGET" ↵
          ><![CDATA[2 1001 1 1000 1]]></BLOB>
      </address>
```

```
      </usage>
    </dataElement>

    <functionElement interfaceKind="export">
      <functionCInterface identifier=          ↵
        "MODULE_IMPL_compute">
        <signature>
          <return>
            <type><void /></type>
          </return>
          <void />
        </signature>
        <fileOrigin name="MODULEM.c" ></fileOrigin>
      </functionCInterface>
      <modelOrigin identifier="Module.compute">
        <name>compute</name>
        <modelLink href="Module.compute" />
        <modelLocation>
          <pathNode name="Module" kind="asd:module">
            <pathParameter name="asd:implementation" ↵
              value="Impl" ></pathParameter>
            <pathParameter name="asd:dataSet"
              value="Data" ></pathParameter>
          </pathNode>
          <pathNode name="compute" kind="asd:process" >
            </pathNode>
        </modelLocation>
        <modelKind kind="process"              ↵
          visibility="public" > </modelKind>
        <runTimeInfo>
          <FPUUsage value="true" ></FPUUsage>
          <TerminateTaskUsage value="false" >   ↵
            </TerminateTaskUsage>
          <messageAccess>
            <message identifier=               ↵
              "MODULE_IMPL_ClassObj.Out1->val" ↵
              send="true" ></message>
          </messageAccess>
          <resourceAccess ></resourceAccess>
          <constraint>
            <period value="0.01" ></period>
            <execution trigger="timer" priority="0" > ↵
              </execution>
            <scheduling mode="preemptive" >    ↵
              <scheduling>
          </constraint>
        </runTimeInfo>
        <annotation>
          <ix:documentation xmlns=             ↵
            "http://www.w3.org/1999/xhtml">
            This is process <i>compute</i> of module ↵
              <i>Module</i>.
          </ix:documentation>
        </annotation>
      </modelOrigin>
    </functionElement>
  </interface>
</module>
```

7 モデリングのヒント

本章では、挙動モデリングツールを用いて INTECRIO 用モデルを作成する際の方法について説明します。

なお本章は、効果的な実行モデルを作成するための「完全な手順」を示すことを目的としているわけではありませんので、モデリングの際の「参考情報」としてお使いください。

7.1 INTECRIO 用のモデリング

INTECRIO に統合するモデルには、ターゲットに依存する情報は含まれません。オペレーティングシステムとハードウェアについてのコンフィギュレーション（プロセスのタスクへの割り当て、シグナルの CAN フレームへのマッピングなど）は、INTECRIO 上で設定します。

INTECRIO へのモデルのインポートは、「ソフトウェアモジュール」を単位として行います。1 つのソフトウェアモジュールに対して 1 つの SCOOP-IX ディスクリプションが必要です。Simulink においてこの「ソフトウェアモジュール」は、任意の数のサブシステムで構成された 1 つの全体モデルに相当します。特殊なケースとして、サブシステム用のコードを生成してこれを 1 つの完全モデルとして扱うこともできます。ASCET の場合は、 1 つの ASCET プロジェクト全体がインポートの単位となります。ASCET プロジェクトは任意の数のクラスとモジュールで構成されます。

INTECRIO には複数のモデルをソフトウェアモジュールとしてインポートできます。各モジュールの入力と出力は、互いに接続したり物理 I/O システムと接続したりでき、必要に応じて動的に、つまり実験実行中にルーティングを変更することができます。ただし、このような接続に使用できるのはスカラ値のみで、ベクトル接続の場合、INTECRIO 内で全体を測定することはできますが、内部のエレメント同士での接続しかできません。

INTECRIO の「システムプロジェクト」は、複数のソフトウェアモジュールとその接続、さらに 1 つのハードウェアシステムと 1 つの OS コンフィギュレーションで構成され、このシステムプロジェクトを単位として実行コード（`*.a21.cod` ファイル）が生成されます。

7.2 Simulink によるモデリング

MATLAB/Simulink Connectivity（5.1 項を参照してください）がサポートしている MATLAB®/Simulink® のバージョンは、R2016a ～ R2021b、および INTECRIO V5.0 のリリース時点においてリリースされているそれらのサービスパックをサポートしています。

サポートされていないバージョンの MATLAB/Simulink では、INTECRIO で使用できるコードを生成できません。

MATLAB/Simulink R2006b ～ R2015b で生成されたコードは、INTECRIO V5.0 にインポートして統合することができます。

MATLAB/Simulink Connectivity がインストールされる際、インストール済みの MATLAB/Simulink の環境が調整され、INTECRIO とのやりとりが行えるようになります。

Simulink と MATLAB® Coder™ + Simulink® Coder™（+ Embedded Coder®）が提供する以下の機能がサポートされています。

- Simulink で使用されるほとんどすべてのブロック

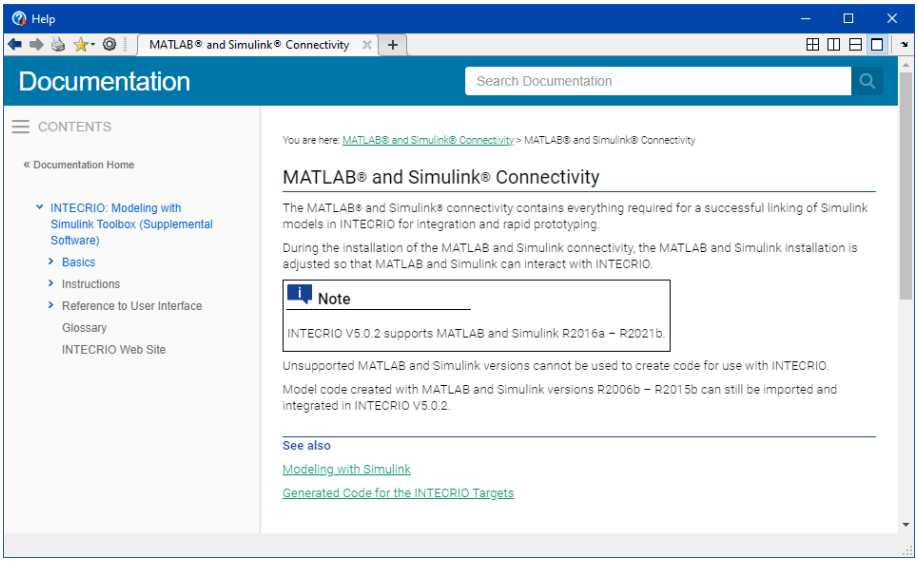
  `To File`、`From File`、`To Workspace`、`From Workspace` というブロックは、ラピッドプロトタイピングターゲットに直接適用できないため、例外です。

  ASAM-MCD-2MC ファイル内に定義されるカーブやマップタイプの適合変数（CHARASTERISTIC）エレメントとして使用できるのは、1-D と 2-D のルックアップテーブルのみです。それより次元数の多いテーブルは使用できません。
- 複数のサンプルレート
- シングルタスキングとマルチタスキング（Simulink のドキュメントを参照してください）
- 連続状態と固定ステップサイズの積分メソッドを含むモデル
- MATLAB Coder + Simulink Coder（+ Embedded Coder）により提供される最適化機構（パラメータのインライニングなど）
- Simulink から供給されるモデル検証ブロック（ランタイムのモデル検証）
- 完全な固定小数点サポート（ランダムなスケーリングによるさまざまなデータ型）
- ユーザー定義 S ファンクション（ノンインライン、ラッパーインライン、フルインライン）
- Stateflow® ダイアグラムのモデル
- 外部モード
- 参照モデル

Simulink における INTECRIO 用のモデリングは、MATLAB Coder + Simulink Coder ／ Embedded Coder によるコード生成を基礎としています。これらに対して INTECRIO 用ターゲット（IRT: INTECRIO Real-Time Target、IER: INTECRIO Embedded Coder Real-Time Target）を提供することにより、INTECRIO の要件に合わせてコード生成を調整することができます。MATLAB/Simulink インターフェースをインストールする際に、IRT または IER へのパスと INTECRIO 固有のブロックへのパスが MATLAB パスのユーザー設定に追加されます。

モデリングは、Simulink、MATLAB Coder、Simulink Coder、Embedded Coder のドキュメントに記述された指示に従って行いますが、ターゲットに依存するサードパーティブロックの使用は避けてください。またコード生成時には IRT または IER をコード生成用ターゲットとして選択してください。

MATLAB/Simulink でのモデリングについて、また MATLAB/Simulink と INTECRIO の併用についての詳細な情報は、MATLAB/Simulink オンラインヘルプの INTECRIO セクションを参照してください。



7.3 ASCET によるモデリング

ASCET プロジェクトは、INTECRIO にインポートされる 1 つの「ユニット」と見なされます。OS コンフィギュレーションの一部は ASCET でも設定することができますが、すべて INTECRIO で行うこともできます。

ASCET によるモデリングの詳細や ASCET と INTECRIO を併用する方法の詳細については、ASCET のオンラインヘルプ（ASCET V6.3 以降）、または INTECRIO-ASC と ASCET-RP のドキュメント（ASCET V6.2 以前）を参照してください。

7.4 ユーザーコードの統合

ユーザーが作成したコードを INTECRIO にインポートする場合は、SCOOP-IX ディスクリプションをマニュアル記述する必要があります。このディスクリプションには、所定のインクルード文、プロセスについてのディスクリプション、測定／適合を行う値についてのディスクリプションなどを含める必要があります。

8 バイパスの概念

8.1 ETK バイパス

ETK を搭載した ECU を使用してバイパス実験を行うには、ECU コードに対して所定の準備を行い、データ構造体や、ラピッドプロトタイピングシステムと ECU 間の通信を設定する必要があります。

またさらに、バイパス手法に起因する安全上の諸問題を考慮する必要があります。

8.2 バイパス入力

測定を行う場合と同様、ECU 変数は、DISTAB13（フックベースバイパスの場合は DISTAB12）のメカニズムを用いてバイパス処理への入力として受け渡されます。

DISTAB13 形式のディスプレイテーブル（**DIS**play **TAB**le）には、各入力変数のアドレスがソートされて格納されます。8、4、2、1 バイトの値がサポートされていて、各アドレスは、ポイントする値のサイズの順で並べられます。ECU ドライバはこのテーブルを解析し、アドレスが指し示す値の内容を ETK RAM 内の戻り値テーブルに書き込みます。戻り値テーブルもアドレスリストと同様、最初にすべての 8 バイト値、続いて 4 バイト値、という順序に並べられます。

この方法により INCA と INTECRIO がマイクロコントローラの内部メモリの値にアクセスすることが可能になります。また、テーブル内のデータをまとめて「ブロックモード」で PC に転送することもできます。

図 8-1 は DISTAB13 のデータレイアウトの概要を示しています。

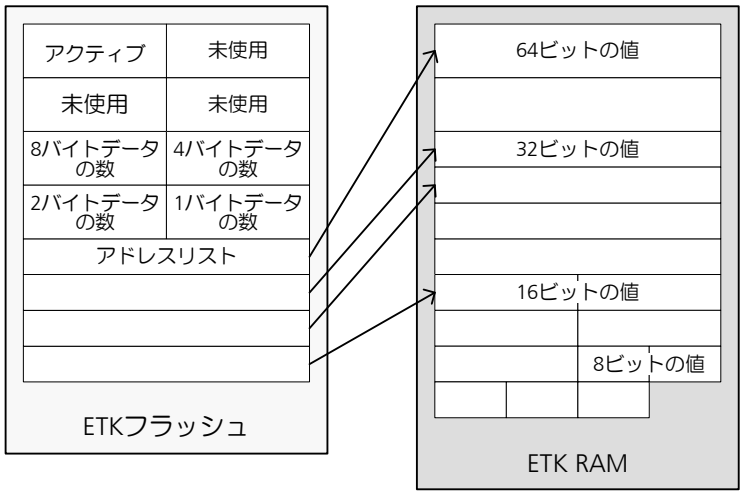


図 8-1　DISTAB13 のデータ構造

「フックベースバイパス」（“hook-based bypass”）の場合、フックを含むバイパスラスタごとに、DISTAB のインスタンスが 1 つ作成され、1 つの DISTAB プロセスが呼び出されます。「サービスベースバイパス」（“service-based bypass”）の場合は、ECU 値をバイパス処理用入力として読み取るサービスを提供／設定するトリガごとに、DISTAB データ構造体のインスタンスが 1 つずつ存在します。

フックベースバイパスの場合、ECU コードに実装される DISTAB の数と名前、およびそれらのサイズ（1 チャンネルあたりのバイト数など）は ECU ソフトウェア内に定義され、A2L ディスクリプションに記述されます。

サービスベースバイパスの場合、使用するメモリ上にすべてのテーブルが動的に割り当てられます。

8.3 フックベースバイパス

従来のタイプ

従来の「フックベースバイパス」においては、バイパスの入力値は測定変数と同じ DISTAB13 メカニズムで収集されます。バイパス入力データが ETK RAM に書き込まれると、バイパス処理がトリガされます。また、バイパス処理の結果を ETK に書き戻すためのチャンネルも用意され、ここに書かれた値を ECU が受け取ります。

各バイパス入力チャンネルに対して出力チャンネルが 1 つずつ提供され、これらのチャンネルのサイズと名前も A2L ディスクリプション内に記述されている必要があります。バイパス実験の設定に応じて、ECU に書き戻す変数の数はプロトタイピングツール（INTECRIO、INTECRIO-RLINK、ASCET-RP）で定義することができます。バイパスにより書き込まれる各変数は ECU ソフトウェア内に用意し、バイパスがアクティブになっている間は、フックを有効にして、ECU がこの値の書き込みを行わないようにする必要があります。「フックコード」は ECU ごと、および変数ごとに異なります。この実装例では、このタスク用のサービスは含まれていません。用意される値は A2L ファイル内に `IF_DATA ASAP1B_BYPASS` として記述されている必要があります。

下図はフックベースバイパスの原理を示しています。オリジナルファンクション（Fn）の結果とバイパスファンクション（Fn*）の結果とをフックで切り替えることができます。

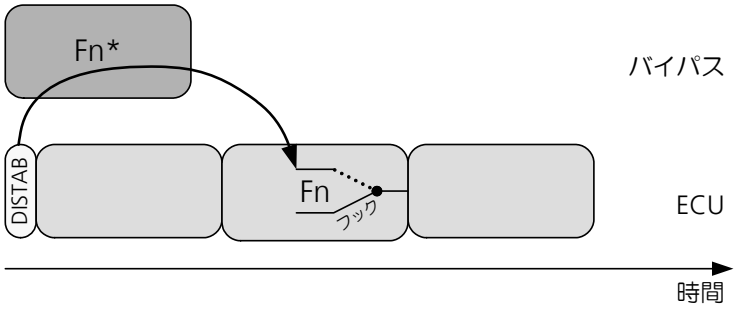


図 8-2　フックベースバイパスの原理

DISTAB17 を使用するタイプ

DIATAB17 を使用することにより、サービスポイントのコンフィギュレーションをフックベースバイパスに拡張することができます。この場合、サービスベースバイパス（SBB）にフックベースバイパス（HBB）が統合されます。つまりフックベースのサービスポイントの場合、ラピッドプロトタイピングシステム内で算出された新しいシグナル値は、サービスポイントの書き込みアクションによってではなく、ECU 関数内の専用のフックコードによって ECU ソフトウェアに転送されます。フックベースのサービスポイントは DISTAB17 以降でサポートされています。

- 以下の図では、**Fn** が ECU 上で実行されるオリジナルの関数を表しています。

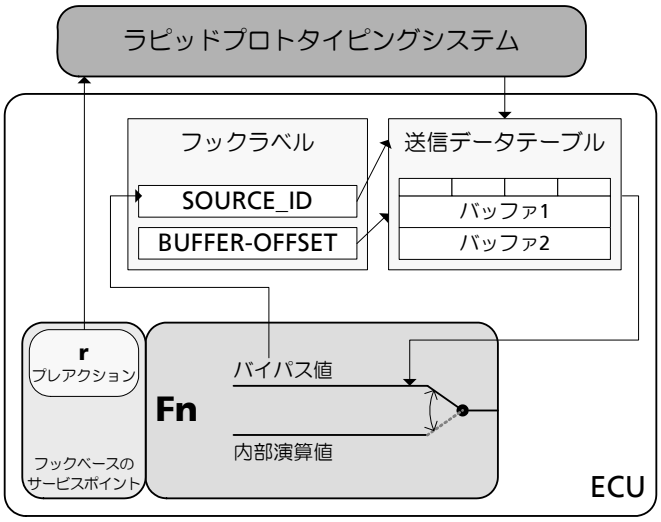


図 8-3フックベースサービスポイントを使用したバイパス

- フックベースのサービスポイントを使用することにより、ECU からラピッドプロトタイピングシステムへ送られるデータを、ECU 上のオリジナル関数が実行される前にプレアクション内で読み取ることができます。フック用のコードは、一般的にオリジナル関数の最後部に実装します。フックは、そのソース（上図の例では SOURCE_ID）とオフセット（上図の例では BUFFER_OFFSET）の情報を、割り当てられたフックラベルから受け取ります。
- ECU 上のオリジナル関数実行中は、ラピッドプロトタイピングシステムはデータをダブルバッファ形式の送信データテーブルに書き込みます。このテーブルはオリジナル関数内にあり、フックによってアクセスされます。
  2 つのバッファは交互に使用されます。ECU 上で実際に使用される値は、バイパス値、またはオリジナル関数による内部演算値のいずれかです。

8.4 サービスベースバイパス

注記

ETK（8 Mbit/s）のサービスベースバイパスはサポートされていません。

「サービスベースバイパス」の場合、バイパスファンクションの入力値と出力値はどちらも同じ DISTAB13 メカニズムにより転送されます。バイパス入力データが ETK RAM に書き込まれると、バイパス処理がトリガされます。また、バイパス処理の結果を ETK に書き戻すためのチャンネルも用意され、これを ECU が受け取りますが、ここではバイパスされる各 ECU プロセスがそれぞれ専用の DISTAB を使用します。

このサービスには、バイパス出力を ECU に書き戻すための「逆方向の」DISTAB メカニズムが含まれます。このサービスは値にバイパス出力を上書きするだけなので、ECU は書き込み先の変数についてフックを使用する必要はありません。INTECRIO は DISTAB のようなテーブルを作成して、書き込み先となる ECU 変数のアドレスをソートして格納し、それに対応する値を ETK フラッシュのテー

ブルに書き込みます。ECU サービス内でバイパス出力を ECU に書き込む部分は、アドレステーブルを解析し、対応する値をデータテーブルから取得して各 ECU アドレスに書き込みます。

下図はサービスベースバイパスの原理を示しています。

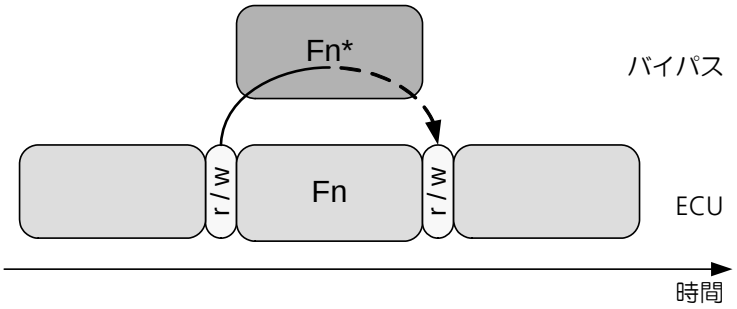


図 8-4　サービスベースバイパスの原理（破線は、バイパスデータの書き戻しを後で行ってもよいことを示しています）

INTECRIO V5.0 は、いくつかのバージョンのサービスベースバイパス（SBB: **s**ervice-**b**ased **bypass**）をサポートしています。表 8-1 に、各ターゲットがサポートしている SBB のバージョンを示し（○：対応、－：未対応）、表 8-2 に各 SBB バージョンで使用できる AML のバージョンを示します。

	SBB V2.0	**SBB V2.1**	**SBB V3.***
ES910.2 / 対応済み ETK	○	○	–
ES910.3 / 対応済み ETK	○	○	○
ES8xx / 対応済み ETK	○	○	○

注記

ETK（8 Mbit/s）のサービスベースバイパスはサポートされていません。

表 8-1　サポートされている SBB のバージョン

SBB バージョン	**使用可能な AML バージョン**		**使用可能な DISTAB タイプ**
V2.0 / V3.0	ETK AML	1.2.0 ～ 1.7.0	DISTAB13 ～ DISTAB16
	XETK AML	1.0.0 以降	
	SBB AML	2.0.0 以降	
V2.1	ETK AML	未対応	DISTAB17
	XETK AML	2.0.0 以降	
	SBB AML	3.1.1 以降	
V3.1	ETK AML	未対応	DISTAB17
	XETK AML	2.0.0 以降	
	SBB AML	3.1.0 以降	

表 8-2　各 SBB バージョンでサポートされている AML と DISTAB のバージョン

8.5　バイパスに関する安全対策

ラピッドプロトタイピングシステム上でバイパスファンクションを実行し、結果のデータを ECU に戻す処理については、その開発や実行を行う際には ECU ソフトウェア本体と同様の配慮が必要です。バイパスソフトウェアの開発や実行の際

に、バイパス出力は ECU の出力チャンネルに直接的または間接的に影響を与える可能性があり、これはラピッドプロトタイピングシステムが専用の出力チャンネルを使用する場合も同様です。

このような観点から、バイパスファンクションにはバイパス出力のレンジチェックと妥当性チェックのアルゴリズムを組み込んでおくことを強くお勧めします。

8.5.1 バイパス入力データ

バイパスを正しく行うためには、整合性のある有効なデータをバイパスに入力しなければなりません。そのためには、ECU ソフトウェアではデータの整合性と妥当性を保証し、不適切または無効な入力を検知した際にはバイパスをアクティブ化しないようにする必要があります。また、ECU ステータスが「初期化中」や「アフターランニング」などの場合、あるいは ECU がエラーモードで稼働している場合も、バイパスをアクティブ化しないようにしなければなりません。このように、バイパスのアクティベーションとバイパスへのデータ転送は、ECU の安全機構が管理する必要があります。

8.5.2 バイパス処理

ECU ソフトウェアは、バイパスがアクティブかどうかを認識し、アクティブであるにもかかわらず実際にバイパス処理が行われなかったりバイパスシステムが予想外に遮断されたり、といったバイパス障害に対応する必要があります。

フェイルセーフの方策としては、バイパスされる ECU ファンクションごとに、その ECU ファンクションの代替出力値を使用したり、代替定数値を使用したり、ECU をリセットしたりすることなどが考えられます。これについてはバイパスドライバを統合する ECU プロバイダが全責任を持つことになります。

サービスベースバイパスの場合、ECU バイパスドライバの実装方法によっては、バイパスされる ECU ファンクションをバイパスユーザーが非アクティブ化することができます。ただし、その場合はバイパスされる ECU ファンクションの結果を代替値として使用できなくなってしまうので、別の安全策を講じる必要があります。

8.5.3 バイパス出力データ

ECU プロバイダは、バイパスシステムからどのような値が戻ってきても ECU の挙動が必ず予測範囲内になるよう保証しなければなりません。バイパス出力値についてはバイパス内で算出される値と同じレンジチェックと妥当性チェックを行う必要があります。

前述したように、ECU ドライバの実装では、あらゆるバイパス障害を ECU が必ず検知できるようにし、有効で安全な代替値が常に使用可能であるようにしておく必要があります。

8.5.4 メッセージコピー

ECU ソフトウェアに「メッセージコピー」の処理が含まれている場合、バイパスシステムがそのことを認識しておく必要があります。ただし、通常の「*フックベースバイパス*」においてはこの限りではなく、フック（および書き込まれるメッセージ）についてはコンパイル前にわかっているので、必要に応じてフックコードがメッセージコピーを使用することができます。つまり各フックにおいて個別のコードとアドレスを使用できます。

「サービスベースバイパス」の場合は、INTECRIO でバイパス実験をセットアップする前にユーザーが変数を選択したり書き込むメッセージを指定したりすることができないので、ECU 内のサービスは汎用的なコードになり、メッセージコピーについて具体的な情報を組み込むことはできません。

この場合、以下の 2 ステップが必要になります。

A ECU ソフトウェアプロバイダはこのメッセージコピー情報を A2L ファイル（通常は暗号化された形式かパスワードで保護された形式のファイル）に記述する必要があります。

B. メッセージコピー情報が暗号化されている場合、バイパスシステムのユーザーは、ECU ソフトウェアプロバイダからパスワードを受け取り、システムのコンフィギュレーション設定時にそれを入力して情報を復号化する必要があります。

上記のいずれか一方でも欠けていると（入力されたパスワードが間違っている場合など）、バイパスシステムは A2L ファイル内の MEASUREMENT 宣言で宣言されているメッセージコピーおよびオリジナルの変数アドレスの読み取り／書き込みについての情報を得ることができません。その結果、receive 変数の場合は古いデータ値を得ることになり、send 変数の場合はバイパスモデルにより書き込まれるデータ値が ECU ソフトウェアの他の部分により上書きされてしまう可能性があり、どちらもバイパスの誤動作の原因になります。

メッセージコピーを含む ECU ソフトウェアには、もう 1 つ重大な問題が発生する可能性があります。ECU タスク用のメッセージコピーを作成するためのオリジナルコードは、所定の用途と適切に生成されたコードをベースにして作られています。しかし変数の値をバイパスメソッドから ECU に書き込むようにすると、データフローが変わり、新規あるいは別のメッセージコピーが必要になる可能性があります。そのため、ECU ソフトウェア内の、バイパスされたファンクションに直接的には関係のないロケーションにおいても、間違った変数値が得られてしまう可能性があります。ある変数をあるロケーション（サービスポイントなど）で書き込むことが危険であるかどうかは、ECU ソフトウェアサプライヤにしか判断できません。この情報を A2L ファイル内で宣言することはできません。

8.6 サービスベースバイパスの特徴

フックベースバイパスにおいてはバイパスされる変数への書き込みを ECU とバイパスのどちらか一方が行いますが、「サービスベースバイパス」では ECU とバイパスの両方が同じ値に連続的に書き込みを行うため、データの不整合が生じる可能性があります。ECU のリアルタイム動作上の制約から、割り込みをディセーブルにすることにより「ECU ファンクションの結果を書き込んでからバイパスの変数値を書き込む」というシーケンスを保護することはできません。

そのため、バイパスサービスを実行するタスクにそれよりも優先度の高いプリエンプティブタスクが割り込みをかけた場合、後者のタスクはバイパスの値ではなく ECU の値を使用してしまう可能性があります。この不整合が発生する可能性は 2 つの書き込みの間隔次第で異なります。

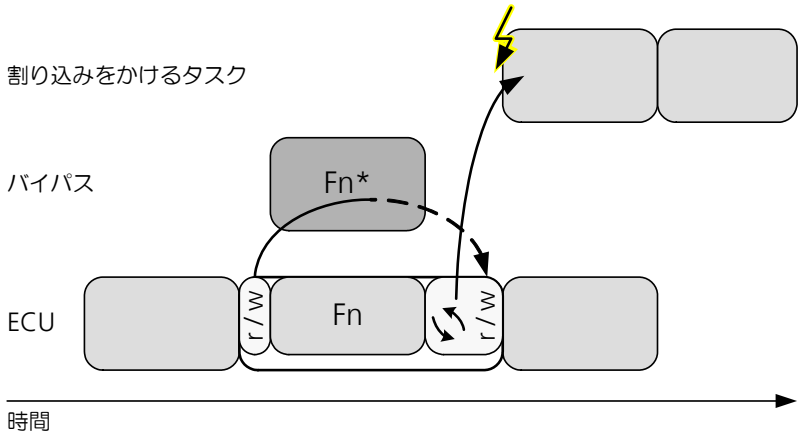


図 8-5　データ不整合の可能性（ はバイパス結果の待ち時間を示します）

対策としては、ECU ファンクションをディセーブルにして当該 ECU 値の書き込みをバイパスだけに行わせる、ということが可能ですが、その場合、バイパス障害に備えてさらに別の安全上の配慮が必要になります（次項を参照してください）。

8.6.1　サービスファンクションとして実装される SBB のためのサービスプロセス

SBB を ECU に実装するには、バイパスされる ECU プロセスを、その ECU プロセスの前後にサービスファンクションコールを行うコンテナプロセスに置き換えます。これにより、当該の ECU プロセスを一定条件下でしか呼び出さないようにすること（たとえばデータの不整合時には非アクティブ化する、など）ができます。ECU プロセスを呼び出さなかった場合は、遅延時間を設けてそのプロセスのタイミング挙動をシミュレートすることができます。

ここで提案した ECU 実装は下図のようになります。

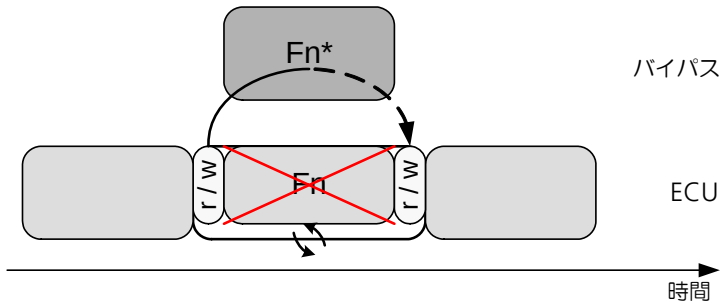


図 8-6　SBB（サービスベースバイパス）の実装例

INTECRIO でバイパス実験をセットアップしてそのサービスポイントを ECU ソフトウェア内にコンテナプロセスとして実装するには、サービスポイントは以下のように定義されます。

A　ECU からデータを受信（プレ読み込みアクション）
B. 送信するデータを待つ（タイムアウト定義済み）
C. ECU にデータを送信（プレ書き込みアクション）

D. オリジナル ECU プロセスを条件付き実行
E. ECU からデータを受信（ポスト読み取りアクション）
F. 送信するデータを待つ（タイムアウト定義済み）
G. ECU にデータを送信（ポスト書き込みアクション）

上記の「プレアクション」と「ポストアクション」は、自由に設定したりアクティブ／非アクティブにしたりすることができます。

8.6.2 ECU の挙動を INTECRIO から制御する

INTECRIO 実験をセットアップした後は、制御変数の初期値を INTECRIO で定義することができます。定義された値は実験初期化時に ETK に書き込まれます。

INTECRIO ユーザーは ECU ファンクションを以下のような方法で制御できます（ただし、ECU ドライバにその機能が備わっている場合のみ）。

- INTECRIO のサービスポイントエディタで ECU ファンクションを非アクティブ化できます（図 8-7 の  列）。
- バイパスエラー検出を定義できます。
- ECU コードのバイパスエラー挙動を調整できます。

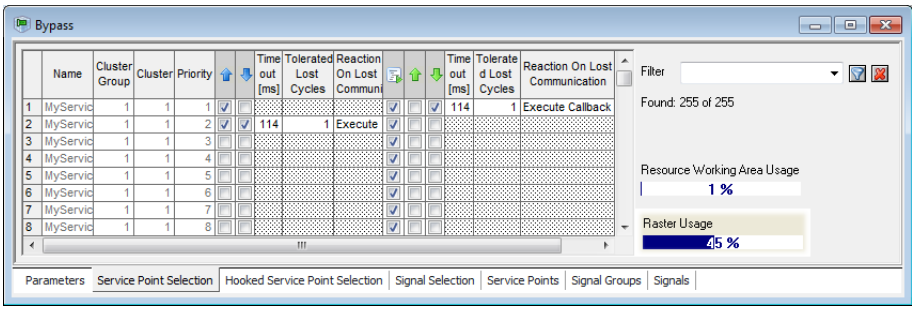


図 8-7　INTECRIO による ECU ファンクション実行制御
（“Cluster Group” ／ “Cluster” 列は SBB V3.* の場合のみ、
“Raster Usage” 列は SBB V2.* の場合のみ表示されます）

8.6.3 サービスベースバイパス V3 用の OS コンフィギュレーション

本項では、サービスベースバイパス V3 を使用するシステムプロジェクトのオペレーティングシステムを設定する方法について説明します。

8.6.3.1 制限事項

サービスベースバイパス V3 を使用するシステムプロジェクトの OS コンフィギュレーションには、以下の制限事項が適用されます。

- 読み取りアクション（受信シグナルグループ  で表されます）は、その読み取りアクションがアクションおよびイベントの両方としてマッピングされる INTECRIO ソフトウェアタスク内でのみ使用できます。

  1 つのイベントは 1 つのタスクにしかマッピングできません。つまり 1 つの読み取りアクションは 1 つの INTECRIO ソフトウェアタスクにしかマッピングできません。
- 読み取りアクションは、タイマタスク、初期化タスク、終了タスク、ISR タスクに割り当てることはできません。
- INTECRIO タスク優先度ごと（つまり ETK トリガごと）に最大 247 個（255 個から内部用に予約されている 8 個を差し引いた個数）の読み取りアクションを使用できます。

- 現行バージョンの RTA-OSEK では、サービスベースバイパスに使用できるタスクの数は最大 253 個に制限されています。これは、使用可能なタスクの総数 256 個のうち、常に 3 個がシステム内部タスクとして使用されるためです。
- 書き込みアクション（送信シグナルグループ で表されます）に関する制限事項はありません。
  - 書き込みアクションは全種類のタスク（ソフトウェア、タイマ、初期化、終了、ISR）にマッピングできます。
  - 1 つの INTECRIO タスクに、複数のサービスポイントクラスタ内の複数のサービスポイントの書き込みアクションを割り当てることができます。

図 8-9 は、複数の ECU タスクから 1 つの読み取りアクションと複数の書き込みアクションが 1 つの INTECRIO タスクに割り当てられている例を示しています。

8.6.3.2 従来の ECU ファンクションバイパス

従来の ECU ファンクションバイパスでは、1 つのサービスポイントにおいてプレ読み取りアクションとポスト書き込みアクションがアクティブ化され、両方のアクションが同じ INTECRIO タスクにマッピングされます。

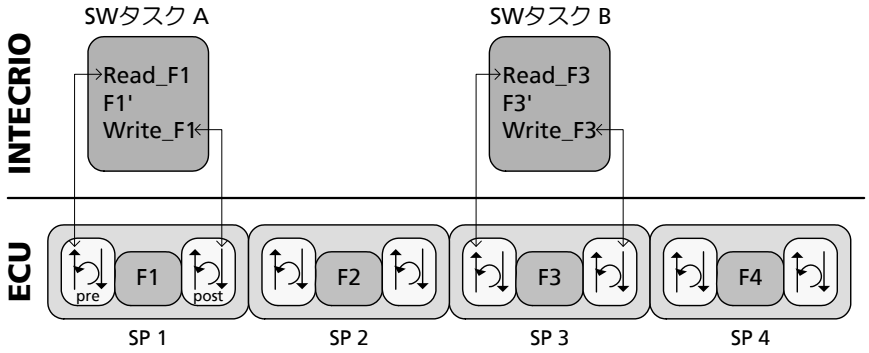


図 8-8　例：従来のバイパス

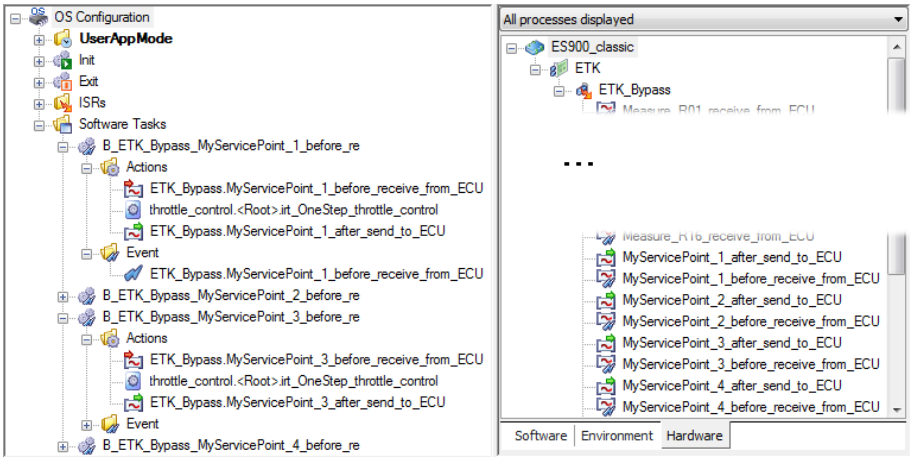
このような OS コンフィギュレーションを実現するには、OSC において以下のように設定します。

A 初めに、OSC の **Auto Mapping** 機能を使用します。

複数のサービスポイントのアクションが複数のソフトウェアタスクにマッピングされ、適切なイベントが割り当てられます。ソフトウェアプロセスは、タイマタスク、初期化タスク、終了タスクのいずれかにマッピングされます。

B. OSC で、当該サービスポイントの ECU ファンクションの代わりとなるソフトウェアプロセスを、そのサービスポイントの読み取りアクションと書き込みアクションが含まれている INTECRIO ソフトウェアタスクに移動します。

C. 必要に応じて、空のタスクを OS コンフィギュレーションから削除します。



従来の ECU ファンクションバイパス用の OS コンフィギュレーション

8.6.3.3 ECU ファンクション全体のバイパス

1 つの INTECRIO タスクに複数の ECU タスクの複数のサービスポイントのプレ書き込みアクションとポスト書き込みアクションを割り当てることができます。これを利用すると、ECU の特定のファンクション（複数の ECU タスクの複数プロセスからなる一連の機能）全体をバイパスできるので、割り込み回数が最小になり、レイテンシを最小化することができます。読み取りシグナルは、ラピッドプロトタイピングターゲット上でタスクが実行されている間は更新されません。ただし他のファンクションの最新値を保証するため、結果の一部はできるだけ早く ECU に送られます。

図 8-9 の例では、サービスポイント SP 1 および SP 2 は高い優先度の ECU タスク A に属し、サービスポイント SP 3 は中程度の優先度の ECU タスク B、サービスポイント SP n は低い優先度の ECU タスク C に属しています。SP 1 の読み取りアクションとすべてのサービスポイントの書き込みアクションが同じ INTECRIO ソフトウェアタスクに割り当てられています。

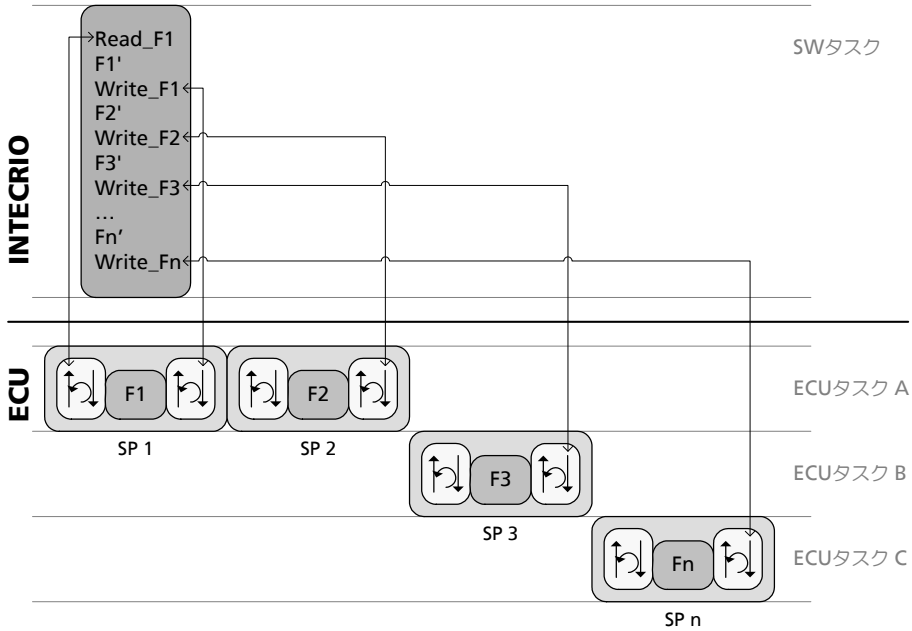


図 8-9　例 : ECU ファンクション全体のバイパス

このような OS コンフィギュレーションを実現するには、OSC において以下のように設定します。

A 初めに、OSC の **Auto Mapping** 機能を使用します。

複数のサービスポイントのアクションが複数のソフトウェアタスクにマッピングされ、適切なイベントが割り当てられます。ソフトウェアプロセスは、タイマタスク、初期化タスク、終了タスクのいずれかにマッピングされます。

B. 自動マッピングにより最速のタイマタスクにマッピングされた書き込みアクション（送信シグナルグループ）を、SP 1 の読み取りアクションが含まれる INTECRIO ソフトウェアタスクに移動します。

OSC の “Hardware” タブで当該シグナルグループを検索し、そのグループのショートカットメニューから **Search** を選択すると、OS コンフィギュレーションツリービュー上でそのグループが強調表示されます。

C. ソフトウェアプロセスを、書き込みアクションと同じ INTECRIO ソフトウェアタスクに移動します。

OSC の “Software” タブで当該プロセスを検索し、そのプロセスのショートカットメニューから **Search** を選択すると、OS コンフィギュレーションツリービュー上でそのプロセスが強調表示されます。

D. 必要に応じて空のタスクを OS コンフィギュレーションから削除します。

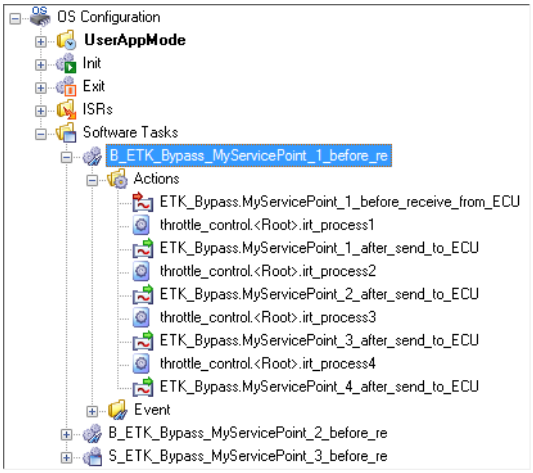


図 8-10 ECU ファンクション全体をバイパスするための OS コンフィギュレーション

8.6.3.4 同じサービスポイントの読み取りアクションと書き込みアクションで異なるラスタを使用する場合

同じサービスポイントの読み取りアクションと書き込みアクションを、それぞれ異なるイベントや周期特性が割り当てられているタスクに割り当てることにより、互いに独立して設定することができます。

図 8-11 は 2 つの例を示しています。左の **(A)** では、優先度が高い方のモデルタスクに、書き込みアクション用の高い方の ETK ラスタ優先度が割り当てられています。右の **(B)** では、優先度が低い方のモデルタスクに、書き込みアクション用の低い方の ETK ラスタ優先度が割り当てられています。

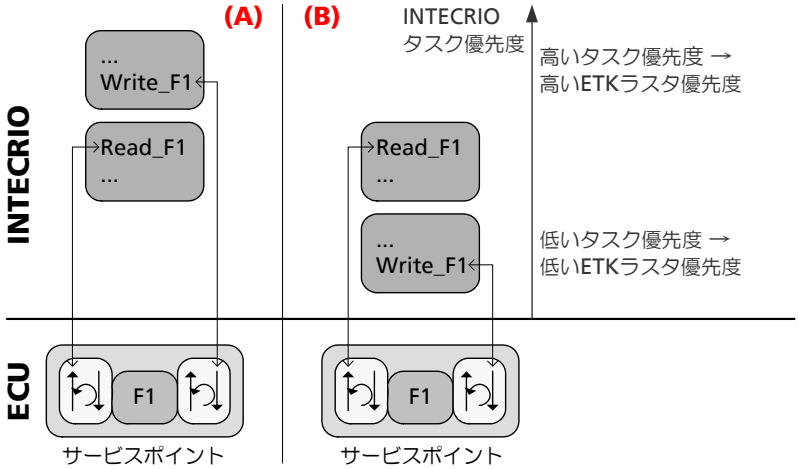


図 8-11　例 : 読み取りアクションと書き込みアクションとでラスタが異なる場合

このような OS コンフィギュレーションを実現するには、OSC において以下のように設定します。

A　当該サービスポイントについて、1 つの読み取りアクションと、その読み取りアクションの後の書き込みアクションをアクティブ化します。

B. マニュアル設定を行う場合 :

- 適切な優先度のソフトウェアタスクを作成し、このタスクの `Actions` フォルダに読み取りアクションを割り当て、同タスクの `Events` フォルダに読み取りアクションのイベントを割り当てます。
- 適切な優先度のタスク（タイマタスクなど）を 1 つ作成し、このタスクの `Actions` フォルダに書き込みアクションを割り当てます。

C. **Auto Mapping** 機能を使用する場合 :

- **Auto Mapping** を実行します。

  1 つのサービスポイントの両アクションが同じソフトウェアタスクにマッピングされます。
- 書き込みアクションを、別の INTECRIO タスクに移動します。

D. ソフトウェアプロセスを適切なタスクにマッピングします。

E. 必要に応じて、空のタスクを OS コンフィギュレーションから削除します。

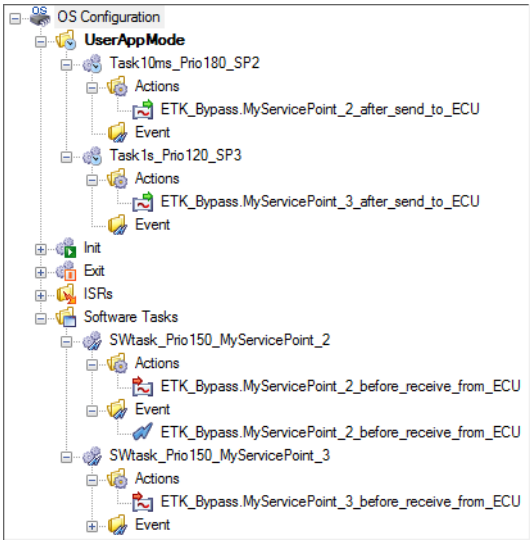


図 8-12　1 つのサービスポイントの読み取りアクションと書き込みアクションが互いに異なるタスクに割り当てられる OS コンフィギュレーション（未完成）

8.6.3.5　ECU 同期ライトバック

「ECU 同期ライトバック」は、ラピッドプロトタイピングシステム（E-Target）上でできる限り早く結果を算出しておき、ECU が適切なタイミングでその結果を得られるようにする仕組みです。

図 8-13 は、サービスポイントを使用して ECU 同期ライトバックを実現する方法を示しています。サービスポイント SP 1 のプレアクション Read_F1 の後、その結果である F3' はラピッドプロトタイピングシステムでは使用可能ですが、ECU 上では使用できません。適切な時点で、サービスポイント SP 3 が F3' を取得し、結果を F3 として ECU に書き込みます。(Read_F3) アクションは ETK チャンネルを解放するためだけに実行される「ダミー」アクションです。このダミーシグナルグループ用にシグナルを選択する必要はありません。

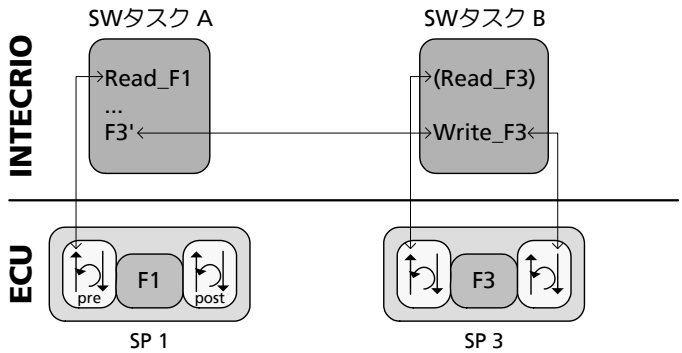


図 8-13　例 : ECU 同期ライトバック

このような OS コンフィギュレーションを実現するには、OSC において以下のように設定します。

A　初めに、OSC の **Auto Mapping** 機能を使用します。

複数のサービスポイントのアクションが複数のソフトウェアタスクにマッピングされ、適切なイベントが割り当てられます。ソフトウェアプロセスはタイマ、初期化、または終了タスクにマッピングされます。

B. ソフトウェアプロセス F3' を、SP 1 の読み取りアクションが割り当てられている INTECRIO ソフトウェアタスクに移動します。

C. 必要に応じて、空のタスクを OS コンフィギュレーションから削除します。

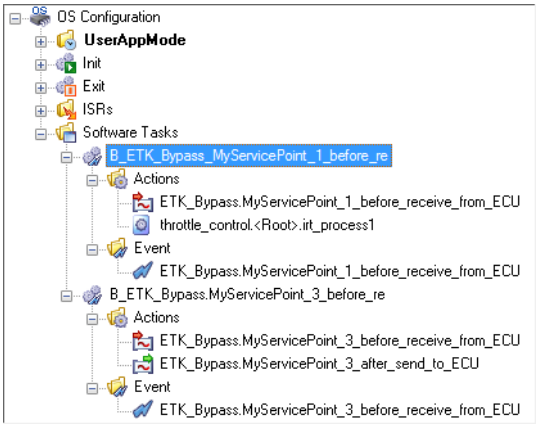


図 8-14　1 つのサービスポイントの読み取りアクションと書き込みアクションが互いに異なるタスクに割り当てられる OS コンフィギュレーション（未完成）

SBB V3 以上では、ビルド処理における OS マッピングおよびトリガ割り当ての結果は、XML ログファイルの形で報告されます。このファイルは *<system name>*_ETKSBBV3_TriggerInfo.xml という名前でワークスペースのログディレクトリに格納されます。

このファイルには、システムプロジェクト内で使用される各サービスポイントアクションについて、シグナルグループが実行されるタスクの名前（TASKNAME） と優先度 （TASKPRIO）、およびシグナルグループデータの転送をトリガする ETK トリガの番号 （TRGNUMBER） と優先度（RASTERPRIO） が記録されます。

例 :

```
<TriggerInfo
 SIGNALGROUPNAME=
    "ETK_Bypass.MySP_1.MySP_1_before_receive_from_ECU"
 RASTERPRIO="8"
 TASKNAME="M_ETK_Bypass_MySP_1_before_re"
 TASKPRIO="195"
 TRGNUMBER="28"
 TRGFLAGADDR="a303efb8"
 RAID="1"/>
```

この情報は、バイパスシステムをデバッグする際に役立ちます。

8.6.4　まとめ

ECU ドライバの実装と統合では以下の点に注意が必要です。

- サービスベースバイパスは ECU 値にバイパス値を上書きするだけで、ECU が値を書き込むことを防ぐことはできません。そのためデータ不整合が発生して予想外の ECU 挙動を招く可能性があります。

- INTECRIO には、損失サイクルの許容最大数を設定するコンフィギュレーション変数があり、たとえば、バイパス値を受け取れなかった ECU 計算サイクルを何サイクルまでエラーと見なさずに許容するかを指定することができます。ただし、バイパス出力値の損失を確実に検知できるかどうかは ECU ソフトウェアの設計に依存します。
- INTECRIO には、具体的なエラー処理を設定するコンフィギュレーション変数があります（ECU にエラー処理が実装されている場合のみ）。ただし、バイパス障害やバイパスの非アクティブ化を ECU ソフトウェアが確実に検知できるかどうかは ECU ソフトウェアの設計に依存します。INTECRIO GUI でのコンフィギュレーション設定により、ECU に実装されている各種エラー処理から適切なものを選択することができます。
- バイパスされる ECU プロセスを INTECRIO で非アクティブ化することができます。ただしそれにより、バイパス障害時に ECU の値を代替値として使用できなくなります。ECU プロセスとバイパスの両方が非アクティブになった場合に意味のあるデータが確実に変数に書き込まれるかどうかは、ECU ソフトウェアの設計に依存します。

9 お問い合わせ先

製品に関するご質問等は、各地域の ETAS 支社までお問い合わせください。

ETAS 本社

ETAS GmbH

Borsigstrasse 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	WWW:	www.etas.com/

日本支社

イータス株式会社

〒 220-6217		
神奈川県横浜市西区	Phone:	(045) 222-0900
みなとみらい 2-3-5	Fax:	(045) 222-0956
クイーンズタワー C 17F	WWW:	www.etas.com/

その他のお問い合わせ先

上記以外のお問い合わせ先につきましては、ETAS ホームページをご覧ください。

各国支社	WWW:	www.etas.com/ja/contact.php
技術サポート	WWW:	www.etas.com/ja/hotlines.php

10 用語集

この用語集では、INTECRIO のマニュアルに使用されている専門用語と略語について解説します。ここに紹介する多くの語は一般的な用語としても用いられていますが、この用語集では、各語が特に INTECRIO について使用される場合の意味が説明されています。

用語はアルファベット順および 50 音順に並んでいます。

10.1 略語

API

Application **P**rogramming **I**nterface（アプリケーション開発者向けプログラミングインターフェース）

ASAM-MCD

自動化および測定システムの標準化を行うための、測定、適合、および診断の作業部会（ドイツ語：**A**rbeitskreis zur **S**tandardisierung von **A**utomations- und **M**ess-systemen, mit den Arbeitsgruppen **M**essen, **C**alibrieren und **D**iagnose）

ASAM-MCD-2MC

ECU ソフトウェアに含まれる適合変数と測定変数についての情報を記述するためのファイルフォーマットで、実験用インターフェースについての情報も保存されます。実験を行う際、このファイル（拡張子は A2L）から実験に必要な情報を読み込みます。

INTECRIO V5.0 は、ETK AML の V1.1（フックベースバイパスのみ）と V1.2 ～ V1.7（フックベースバイパス／サービスベースバイパス）、XETK AML の V2.5 まで、ASAP1B_Bypass AML の V1.0 以降、SBB AML の V2.0、V3.0、V3.1 をサポートしています。

詳しい情報は https://www.asam.net/ を参照してください。

ASCET

ECU ソフトウェア開発用の ETAS 製品ファミリ — ASCET のモデルを INTECRIO にインポートできます。

ASCET-MD

ASCET Modeling & **D**esign（ASCET 製品ファミリに含まれる BMT）

ASCET-RP

ASCET Rapid **P**rototyping（ASCET 製品ファミリのラピッドプロトタイピングツール）

AUTOSAR

Automotive **O**pen **S**ystem **A**rchitecture（自動車用オープンシステムアーキテクチャ）。https://www.autosar.org/ を参照してください

BMT

Behavioral **M**odeling **T**ool（挙動モデリングツール）— BMT では、モデルの挙動を編集し、シミュレーションやアニメーションを行い、ファンクションコードを生成することができます。

BR_XETK

車載イーサネットインターフェースを採用した ETK（エミュレータテストプローブ）

BR_XETK を使用するには、ES882 ／ ES886 を含む ES800 ハードウェアシステムが必要です。

BSW

Basic **s**oft**w**are（基本ソフトウェア）— 通信や I/O の機能のほか、各種ソフトウェアコンポーネントに必要な諸機能を提供します。

CAN

Controller **A**rea **N**etwork（コントロールエリアネットワーク）— 強靭な車両バス規格のひとつ。ホストコンピュータがなくてもマイクロコントローラと各種デバイスのアプリケーション同士が互いに通信し合うことができます。

CANdb

CAN **d**ata**b**ase — Vector Infomatik 社の CANdb データ管理プログラムで作成された CAN ディスクリプションファイル

INTECRIO V5.0 は CANdb V2.3 以降をサポートしています。

CPU

Central **P**rocessing **U**nit（中央演算処理装置）— INTECRIO では、マイクロコントローラを意味します。

DISTAB

Display **Tabl**e（ディスプレイテーブル）— ETK バイパス実験において実験ターゲットと ECU との間のデータ交換に使用されるデータ交換メソッド

INTECRIO V5.0 は、DISTAB12 以降（XETK AML：DISTAB13）を用いたフックベースバイパスと、DISTAB13 を用いたサービスベースバイパスをサポートしています。

INTECRIO V5.0 では、ES830 または ES910.3 を使用することにより DISTAB17 を用いたバイパスも利用可能です。

ECU

Electronic **C**ontrol **U**nit（電子制御ユニット）— CPU と周辺機器からなる、小型の組み込み型コンピュータシステム

一般的に、すべてのコンポーネントが 1 つの筐体に収められています。

ETK

Emulator-**T**as**k**opf（「エミュレータテストプローブ」を表すドイツ語）

FETK

ES89x ECU ／バスインターフェースモジュール対応の ETK

FIBEX

Field **B**us **Ex**change — XML スキーマをベースとするデータ交換フォーマットで、車載通信ネットワーク全体を記述するものです。

FIBEX はさまざまなネットワークタイプ（CAN、LIN、MOST、FlexRay）用に定義されていて、バスアーキテクチャ、信号、ネットワークノードのプロパティなどについての情報を記述できるようになっています。

FIBEX ファイルフォーマットは ASAM（Association for Standardization of Automation and Measuring Systems：自動化／測定システム標準化委員会）により標準化されています。

INTECRIO V5.0 は FIBEX ベースラインバージョンの FIBEX V2.0.x と V3.1.0（V3.1.0 については制限事項がありますので、詳しくはオンラインヘルプを参照してください）をサポートしています。

詳細については https://www.asam.net/ を参照してください。

FIFO

First **i**n、**f**irst **o**ut（先入れ先出し）

HBB

hook-**b**ased **b**ypass（フックベースバイパス）

HC

Hardware **C**onfigurator（ハードウェアコンフィギュレータ）

IER

INTECRIO **E**mbedded Coder **R**eal-Time Target（INTECRIO に Simulink モデルをインポートするための Embedded Coder リアルタイムターゲット）

INCA

Integrated **C**alibration and **A**cquisition Systems（ETAS の測定／適合システム）

INTECRIO V5.0.2 には INCA V7.2.17 以降が対応しています。

INCA-EIP

INCA のアドオンです。INCA からラピッドプロトタイピングターゲット（ES910、ES830）や仮想プロトタイピングターゲット（VP-PC）へのアクセスを実現します。

INTECRIO V5.0.2 には INCA-EIP V7.2.17 以降が対応しています。

INTECRIO-RP

INTECRIO **R**apid **P**rototyping パッケージ — ラピッドプロトタイピングへの接続機能を提供します。

INTECRIO-VP

INTECRIO **V**irtual **P**rototyping パッケージ — 仮想プロトタイピングへの接続機能を提供します。

IRT

INTECRIO **R**eal-Time **T**arget（INTECRIO に Simulink モデルをインポートするための Simulink Coder リアルタイムターゲット）

LDF

LIN **d**escription **F**ile（LIN ディスクリプションファイル）— LIN コントローラ用コンフィギュレーションファイル

INTECRIO V5.0.2 は LDF V1.3、V2.0、V2.1、V2.2 をサポートしています。

LIN

Local **I**nterconnect **N**etwork（ローカル相互接続ネットワーク）— 車載コンポーネント間の相互通信に使用されるシリアルネットワークプロトコル

CAN ほどの帯域幅や汎用性を必要としない用途において使用され、一般的な例としては、電気自動車のドアやシートのネットワーク接続などが挙げられます。

LSB および lsb

大文字表記の場合は最下位バイト（**L**east **S**ignificant **B**yte）— 小文字表記の場合は最下位ビット（**l**east **s**ignificant **b**it）

MDA

Measure **D**ata **A**nalyzer（測定データアナライザ）— 保存された測定データを表示して分析するための ETAS オフラインツール

MSB および msb

大文字表記の場合は最上位バイト（**M**ost **S**ignificant **B**yte）— 小文字表記の場合は最上位ビット（**m**ost **s**ignificant **b**it）

OIL

OSEK **I**mplementation **L**anguage（OSEK 実装言語）— ECU ネットワーク用記述言語

OIL はオペレーティングシステムを間接的に構成するパーツで、ECU ネットワークの静的情報（通信接続や ECU のプロパティなど）の記述に使用されます。

OS

Operating **S**ystem（オペレーティングシステム）

OSC

Operating **S**ystem **C**onfigurator（OS コンフィギュレータ）

OSEK

Offene **S**ysteme für die **E**lektronik im **K**raftfahrzeug（自動車エレクトロニクス向けオープンシステムのための作業部会のドイツ語表記）

PDU

Protocol **d**ata **u**nit（プロトコルデータユニット）— プロトコルスタック内のレイヤ間で受け渡しされる情報の単位で、ペイロードと制御情報が含まれます。

INTECRIO V5.0 では、FlexRay の PDU はシグナルグループに相当します。

RE

Runnable Entity（ランナブルエンティティ） — 実行時において RTE からトリガされる一連のコード。INTECRIO で扱われる従来の「プロセス」に相当します。

RP

Rapid **p**rototyping(ラピッドプロトタイピング):164 ページ「ラピッドプロトタイピング」を参照してください。

RTA-OSEK

ETAS のリアルタイムオペレーティングシステム — AUTOSAR-OS V1.0（SC-1）と OSEK/VDX OS V2.2.3 を実装し、MISRA コンポーネントに完全準拠しています。

RTA-OS

ETAS のリアルタイムオペレーティングシステム — AUTOSAR R3.0 OS と OSEK/VDX OS V2.2.3 を実装し、MISRA コンポーネントに完全準拠しています。

RTA-RTE

ETAS が提供する AUTOSAR ランタイム環境

RTE

AUTOSAR **r**un**t**ime **e**nvironment（AUTOSAR ランタイム環境）— SWC（ソフトウェアコンポーネント）、BSW（基本ソフトウェア）、OS（オペレーティングシステム）間のインターフェースとなるものです。

RTIO

Real-**T**ime **I**nput-**O**utput（リアルタイム I/O）

SBB

Service-**b**ased **b**ypass（サービスベースドバイパス）

SBC

Sensotronic **B**rake **C**ontrol（電気油圧式ブレーキシステム）

SCOOP

Source **Co**de, **O**bjects, and **P**hysics（ソースコード、オブジェクト、および物理的現象）

SCOOP-IX

SCOOP **I**nterface E**x**change language（SCOOP インターフェース交換言語）

INTECRIO V5.0 は SCOOP-IX の V1.0、V1.1、V1.2、V1.4、V1.5 をサポートしています。

SP

Service **p**oint：163 ページ「サービスポイント」を参照してください。

SWC

AUTOSAR **S**oft**w**are **C**omponent（AUTOSAR ソフトウェアコンポーネント）— AUTOSAR アプリケーションにおけるソフトウェアの最小単位

UDP

User Datagram Protocol

UML

Unified **M**odeling **L**anguage（統一モデリング言語）

VFB

AUTOSAR の **V**irtual **f**unction **b**us（仮想ファンクションバス）

VP

Virtual **p**rototyping （仮想プロトタイピング）:162 ページ「仮想プロトタイピング」を参照してください。

XCP

Universal measurement and calibration protocol（汎用測定／適合プロトコル）— x は、さまざまな種類の転送層を使用できることを表します。正式名称は ASAM MCD-1 XCP です。

INTECRIO V5.0 は XCP V1.0 のほか、V1.0 に互換なすべての上位バージョンをサポートしています。さらに、V1.1 以降の `XCPplus` キーワードもサポートしています。

XETK

イーサネットインターフェースを採用した ETK（エミュレータテストプローブ）

XML

E**x**tensible **M**arkup **L**anguage（拡張可能なマークアップ言語）

10.2 用語

AUTOSAR ソフトウェアコンポーネント

「SWC」を参照してください。

Embedded Coder™

Simulink® Coder™ 用アドオン。Simulink Coder の機能を拡張して、組み込み型ターゲット上での量産アプリケーションの記述、統合、デプロイ、およびテストを可能にします。

FlexRay

FlexRay は高い伝送速度で決定論的データ交換を行うための、拡張可能な耐障害性の通信システム。時分割の手続きにより、モジュール性に優れ安全性の高い分散システムを構築でき、2 チャンネル上で 10MBaud の高周波数帯域を実現します。現代の自動車は革新的な電子システムの搭載量が年々多くなり、ネットワーク負荷も増大しつつありますが、高周波数帯域の使用によって、このような状況への対応も可能となります。

この通信システムの仕様は、世界中の自動車メーカーとサプライヤに広くサポートされている「FlexRay コンソーシアム」という委員会により策定されました。

INTECRIO

各種 BMT（挙動モデリングツール）で作成された制御アルゴリズムのパーツを統合し、プロトタイピング実験の環境を構築するツール。ハードウェアシステムのコンフィギュレーションを設定し、実験用ハードウェアと制御アルゴリズムを接続することができます。

MATLAB®

技術計算用の高性能言語。1 つの環境に、数学計算、視覚化、プログラミングなどの機能が備わっています。

MATLAB® Coder™

MATLABコ ード用のコードジェネレータ

OS コンフィギュレータおよび OSC

オペレーティングシステムに含まれるタスクは、INTECRIO 内の **OS コンフィギュレータ**によって設定されます。**OSC** エディタは、この設定を行うためのエディタです。このエディタでは、システム構成を素早く把握でき、アプリケーションからの視点で表示されたソフトウェアの要素を、容易にオペレーティングシステムに組み込むことができます。

RTA-Trace

ECU への汎用インターフェースを介してシステム挙動を監視できるソフトウェアトレースツール。現在、生産中止していますが、既存のインストーラーは現在も使用できます。

Simulink®

動的システムのモデリング、シミュレーション、分析を行うためのツール。Simulink のモデルを INTECRIO にインポートできます。

Simulink® Coder™

Simulink ／ Stateflow モデル用のコードジェネレータ。MATLAB® Coder™ が必要です。

Stateflow®

複雑なイベントドリブンシステムのモデリングとシミュレーションを行うためのツール。MATLAB/Simulink にシームレスに統合されています。

X パス実験

バイパス実験とフルパス実験を組み合わせた実験。実験ターゲット（ES900、ES800）は、バイパスフックを持つ ECU を外部インターフェースとして利用します。

アクチュエータ

実行ハードウェアユニット。電子信号処理とメカニクスとの物理的なインターフェースです。

アプリケーションモード

アプリケーションモードはオペレーティングシステムの実行モードです。EEPROM プログラミングモード、初期化モード、通常運転モードなど、システムのさまざまな状態に対応します。

イベント

オペレーティングシステムのアクション（タスクなど）を開始するための外部トリガ。

イベントインターフェース

「プロセス」を参照してください。

インプリメンテーション

物理的に定義されたモデル値から、実際に実行されるプログラム用の固定小数点値への変換規則を記述したもの。1 つの線形変換式とモデル値の範囲制限とで定義されます。

仮想プロトタイピング

自動車の電子制御ファンクションの仮想プロトタイプを作成することにより、PC 上で制御ロジックのテストを行えます。

環境システム

環境システムは、仮想プロトタイピングにおいてプラントモデルをモデリングするためのものです。ソフトウェアシステムと同様、モジュールとファンクションで構成され、作成方法もソフトウェアシステムの場合と同じです。

基本ソフトウェア

「BSW」を参照してください。

グラフィカルフレームワーク

INTECRIO のメインウィンドウ。このウィンドウでさまざまな INTECRIO コンポーネントを統合します。

クロスバー

AUTOSAR に該当しないモジュール、ファンクション、ハードウェアの間で行われる通信について、その管理と制御を行うものです。

検証

ある開発フェーズの成果が、そのフェーズの仕様を満たしているかどうかを判断するために、システムまたはコンポーネントを評価するプロセス。ソフトウェアの検証においては、特定の開発ステップについて定義された仕様がソフトウェアに正しく実装されているかを検証します。

コネクション（静的）

静的コネクションとして定義されたシグナルソースとシグナルシンク間の接続は、ランタイムにおいて変更できません。

コネクション（動的）

動的コネクションとして定義されたシグナルソースとシグナルシンク間の接続は、ビルド処理を行わなくてもランタイムにおいて変更できます。

サービスポイント

ECU ソフトウェア内のプロセスをカプセル化したもので、ターゲットシステムとのデータ転送アクションを行います。これらのアクションはユーザーが有効化したり設定したりすることができます。

サービスポイントクラスタ

ECU において同じ優先度で実行される複数のサービスポイントのグループ（同じ ECU タスク内に配置された複数のサービスポイント）

サービスポイントクラスタグループ

複数のサービスポイントクラスタのグループ。グループには、ECU 内で同時に呼び出される可能性のあるすべてのタスクのすべてのサービスポイントが含まれます。

システムプロジェクト

ハードウェアシステム、ソフトウェアシステム、環境システム（仮想プロトタイピングの場合）、シグナルのマッピング、およびオペレーティングシステムのコンフィギュレーションを、1 つのプロジェクトとして一体化したもの。実行コードは、システムプロジェクト単位で生成されます。

センサ

センサは、物理的または化学的な量（通常は電気的でない量）を電気的な量に変換します。

ソフトウェアシステム

ソフトウェアシステムには ECU ファンクションの汎用部分（モジュール、ファンクション、接続情報）が含まれています。

タスク

タスクは複数のプロセスの実行シーケンスを定義するもので、オペレーティングシステムにより起動されます。タスクの属性には、アプリケーションモード、起動トリガ、優先度、スケジューリングモードなどがあります。タスクが起動されると、タスクに割り当てられた一連のプロセスが、指定の順序で実行されます。

妥当性確認

作成されたシステムまたはコンポーネントが、アプリケーションに対するユーザー要件を満たすものであるかどうかを判断するための評価プロセス。ここでは、ユーザーの承認がとれるシステムまたはコンポーネントが作成されたかどうかを確認します。

統合

複数の会社においてさまざまなツールを用いて開発されたモデルコードを結合して制御アルゴリズムを構築し、ターゲットハードウェア用にそのアルゴリズムを調整して、最終的に実行ファイルを作成すること。

ハードウェアシステム

ハードウェアシステムは実験に使用するハードウェア全体を定義するもので、ECU（実験ターゲット）についての定義や ECU 間インターフェース（バスシステム）の記述などで構成されます。

バイパス実験

バイパス実験においては、ECU プログラムの一部が実験ターゲット（ES900、ES800）上で実行されます。バイパス実験を行うためには、ECU プログラム内に専用のフックルーチンを挿入する必要があります。

INTECRIO V5.0 は、CAN ／ UDP を用いた XCP バイパスや、フックドベース／サービスベースの ETK ／ XETK ／ FETK バイパスをサポートしています。

ファンクション

ソフトウェアシステムをグルーピングするためのアイテムで、ファンクション自体に機能はありません。1 つのファンクション内には複数のモジュールが組み込まれ、接続されています。関連するモジュールをファンクションにまとめることにより、モデル機能を明確に把握でき、再利用もしやすくなります。

フルパス実験

フルパス実験においては、ECU プログラム全体が実験ターゲット上で実行されます。

プロジェクトインテグレータ

プロジェクトインテグレータは、システムのすべてのコンポーネント（モジュールとファンクション、ハードウェアインターフェース、OS コンフィギュレーションなど）を結合して 1 つの実行ファイルを生成します。

プロジェクトコンフィギュレータ

INTECRIO の統合プラットフォームの一部。ソフトウェアシステムとシステムプロジェクトを設定する際に使用されます。

プロセス

オペレーティングシステムにより起動され、並行的に実行できる機能単位。プロセスはモジュール内に記述し、引数（入力）や戻り値（出力）を持ちません。

プロセッサ

「CPU」を参照してください。

プロトタイプ

実験ターゲットシステム用の実行ファイル。プロトタイプには ECU ファンクションが実行可能なコードとして書き込まれていますが、その構成と目的は、最終的な製品用のものとは異なります。

モジュール

INTECRIO のモジュールは、ECU ファンクションが汎用的に記述されたものです。1 つのモジュールは、1 つの ASCET プロジェクトや Simulink モデルに相当します。

ラピッドプロトタイピング

実験ターゲット（車両インターフェースを持つコンピュータ）上でソフトウェアを実行して、制御アルゴリズムの検証と妥当性評価を行うこと。

ランタイム環境

「RTE」を参照してください。

ランナブルエンティティ

「RE」を参照してください。

レガシー AUTOSAR モジュール（または **レガシー SWC**）

V5.0.0 以前の INTECRIO にインポートされた AUTOSAR モジュール

ワークスペース

ワークスペースにおいて、INTECRIO での作業で生成されたすべてのデータが管理され、結合されます。WS ブラウザ（ワークスペースブラウザ）のツリービューから、INTECRIO のすべてのアイテムを開くことができます。

図

表

索引